Lose Sammlung von Beiträgen

aus

der Zeitschrift

ComputerKurs

der Jahrgänge 1984/1985

Boolesche Algebra

Die Funktionsweise des Computers basiert auf der Abfolge von „High"- und „Low"-Zuständen in seinen Schaltkreisen. Damit der Rechner die Impulsfolgen weiterverarbeiten kann – also etwa eine Addition ausführen – sind spezielle Logikschaltungen nötig.

Theoretische Basis jedes Computers ist die Boolesche Algebra – eine mathematische Methode, die mit wenigen einfachen Regeln und Sätzen wahre von falschen Aussagen trennt.

Am Anfang dieses Kurses steht eine detaillierte Betrachtung der theoretischen und praktischen Merkmale logischer Schaltungen, die mit Beispielen aus der Technik Ihres eigenen Computers ergänzt werden. In der Booleschen Algebra gibt es drei Grundbegriffe: AND, OR und NOT (und, oder, nicht). Ihre Funktion kommt dem üblichen Sinn der Worte sehr nahe, wie dieses Beispiel verdeutlicht:

Wenn es Samstag ist AND die Sonne scheint, geht Thomas spazieren.

Ob Thomas spazierengeht, hängt also von zwei Dingen ab: Die Sonne muß scheinen, und es muß Samstag sein. Wenn er sich entscheidet, muß Thomas nur wissen, ob die Aussagen „die Sonne scheint" und „es ist Samstag" richtig oder falsch sind. Vier verschiedene Kombinationsmöglichkeiten dieser Eigenschaften gibt es, aber nur bei einer wird Thomas spazierengehen. Die verschiedenen Kombinationen von wahren und falschen Aussagen lassen sich auf einer Wahrheitstabelle darstellen:

DIE SONNE SCHEINT	ES IST SAMSTAG	THOMAS GEHT SPAZIEREN
FALSCH	FALSCH	FALSCH
FALSCH	WAHR	FALSCH
WAHR	FALSCH	FALSCH
WAHR	WAHR	WAHR

Die logische OR-Funktion läßt sich ähnlich darstellen. Betrachten Sie das Beispiel:

Udo geht essen, wenn Kai OR Eva mitkommt.

Wieder haben wir zwei Bedingungen, von denen eine Handlung abhängt: Entweder begleitet ihn Kai oder Eva. Wie bei der AND-Funktion ergibt sich eine Wahrheitstabelle, die natürlich auch vier Kombinationsmöglichkeiten (zwei Bedingungen, jede entweder wahr oder falsch) darstellt:

KAI KOMMT MIT	EVA KOMMT MIT	UDO GEHT ESSEN
FALSCH	FALSCH	FALSCH
FALSCH	WAHR	WAHR
WAHR	FALSCH	WAHR
WAHR	WAHR	WAHR

Noch einfacher ist es bei NOT, der dritten logischen Funktion. Beispiel:

Ich gehe aus, wenn es NOT dunkel ist.

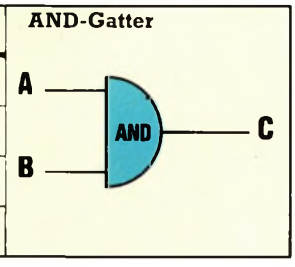
Hier ist nur eine Bedingung zu beachten, ob es dunkel ist oder nicht. Die Wahrheitstabelle zeigt daher bei diesem Beispiel nur zwei Kombinationsmöglichkeiten:

ES IST DUNKEL	ICH GEHE AUS
FALSCH	WAHR
WAHR	FALSCH

Die Schaltung eines Computers ist aus elektronischen Elementen aufgebaut, die den drei logischen Grundfunktionen AND, OR und NOT entsprechen. Eine wahre Bedingung wird dabei durch die Binärziffer 1, eine falsche Bedingung durch die Binärziffer 0 dargestellt. Man kann daher für jedes logische „Gatter" eine Wertetabelle aufstellen, die für eine Kombination von Eingangswerten den entsprechenden Ausgangswert verzeichnet. Die Namen der logischen Gatter entsprechen der Bezeichnung ihrer Funktion in der Booleschen Algebra.

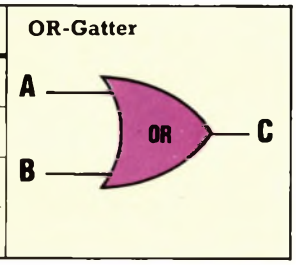
Diagramm und Wertetabelle eines AND-Gatters mit Eingängen A, B und Ausgang C:

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



In Worten ist die Funktion des AND-Gatters so zu beschreiben: Der Ausgang ist 1, wenn beide Eingänge 1 sind, sonst ist er 0. Die Boolesche Schreibweise für die Ausgabe des Gatters: A.B. Wahrheitstabelle und Diagramm des OR-Gatters:

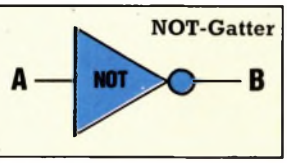
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1



Die Funktion des OR-Gatters läßt sich durch den Satz beschreiben: Der Ausgang ist 1, wenn einer oder beide Eingänge 1 sind. Der Boolesche Ausdruck dafür lautet A+B.

Das NOT-Gatter hat, anders als das AND- bzw. OR-Gatter, nur einen Ein- und einen Ausgang.

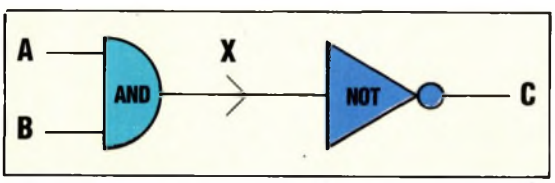
A	B
0	1
1	0



Beim NOT-Gatter zeigt der Ausgang immer den umgekehrten Wert des Einganges. Die Boolesche Schreibweise für die Ausgabe ist $\overline{A}$.

Verknüpfung logischer Gatter

In einem Computer sind eine Vielzahl logischer Gatter zu einem komplexen Schaltkreis verbunden oder „vernetzt". Auch die Funktion vernetzter logischer Gatter ist durch eine Wahrheitstabelle zu beschreiben, die jeder möglichen Eingabe die entsprechende Ausgabe zuordnet. Ein einfaches Beispiel:

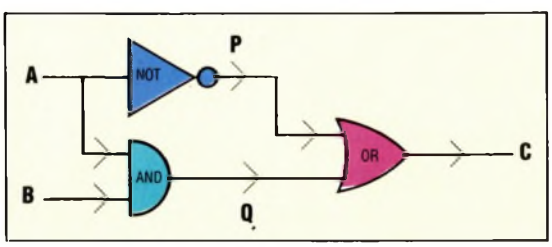


Die Schaltung hat die Eingange A und B sowie den Ausgang C. Den Ausgang des ersten Gatters haben wir mit X bezeichnet, damit sich die Wertetabelle leichter aufstellen laßt. Durch die zwei Eingange sind vier Kombinationen der Eingangsvariablen möglich

A	B	X	C
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

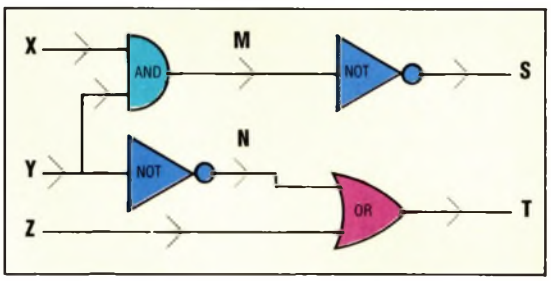
Der Ausgang des AND-Gatters führt über das NOT-Gatter zum „Ergebnis": Ausgang C.

Die nachste Schaltung ist schwieriger, obwohl es wieder nur zwei Eingange und damit vier Zeilen in der Tabelle gibt. Rechts in der Wahrheitstabelle (P, Q und C) findet sich die umgestellte OR-Tabelle wieder.



A	B	P	Q	C
0	0	1	0	1
0	1	1	0	1
1	0	0	0	0
1	1	0	1	1

Eine Wahrheitstabelle läßt sich auch für Schaltungen mit mehr als zwei Eingängen und einem Ausgang aufstellen. Unten ein Beispiel für drei Ein- und zwei Ausgänge:

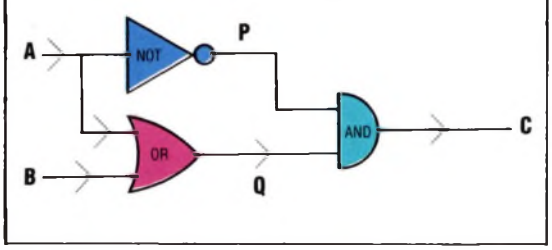


Bei drei Eingängen gibt es 2*2*2=8 mögliche Kombinationen für die Eingabe:

X	Y	Z	M	N	S	T
0	0	0	0	1	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	0	0	0
1	1	1	1	0	0	1

UBUNG 1

1. Schreiben Sie die Tabelle zu dieser Aussage: „Wenn er den Führerschein hat OR wenn ein Fahrlehrer dabei ist, darf Max fahren."
2. Welche Wahrheitstabelle gehört zu dieser Aussage: „Ein Programm kann geladen werden, wenn ein Cassettenrecorder OR ein Diskettenlaufwerk vorhanden ist AND das Programm NOT für einen anderen Rechner geschrieben wurde"?
3. Ermitteln Sie die Wahrheitstabelle für die unten abgebildete Schaltung:

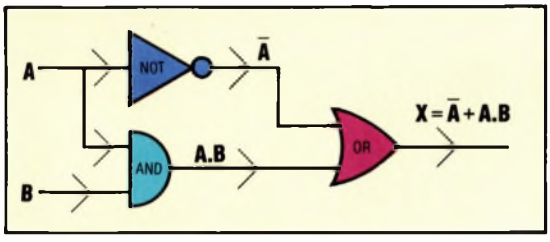


Rechner-Addition

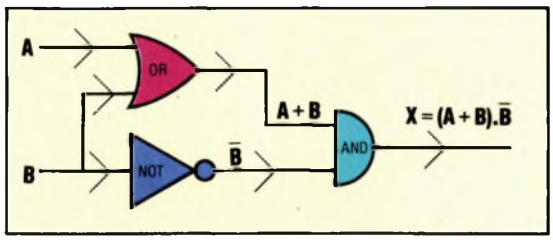
Im ersten Abschnitt des Kurses haben wir die drei grundlegenden logischen Operationen (AND, OR und NOT) sowie das Prinzip einfacher Logikschaltungen kennengelernt. Danach ist es nun leicht zu verstehen, wie der Computer eine Addition ausführt.

Die algebraische Behandlung logischer Beziehungen wird nach dem Mathematiker George Boole (1815–1864) „Boolesche Algebra" genannt. Im Computerbereich ist die Boolesche Algebra unverzichtbar, weil sie die mathematische Vereinfachung logischer Schaltungen ermöglicht. In der Praxis bedeutet das: weniger Gatterbausteine für eine gegebene Funktion – und damit eine höhere Arbeitsgeschwindigkeit.

Die Boolesche Symbolik für die Ausgabe der drei wichtigsten logischen Gatter kennen Sie bereits: AND (A.B), OR (A+B) und NOT ($\overline{A}$). Kompliziertere Schaltungen entstehen durch Zusammensetzung dieser Ausdrücke. $X=\overline{A}+A.B$ entspricht zum Beispiel dieser Schaltung:



Wichtig ist die Reihenfolge, in der die Operationen AND und OR ausgeführt werden. Dabei gilt die Regel AND vor OR (und vor NOT). Diese Reihenfolge läßt sich mit Hilfe von Klammern ändern. Beispiel: $X=(A+B).\overline{B}$. In diesem Ausdruck werden A OR B mit dem entgegengesetzten Wert von B durch AND verknüpft. Hier die Schaltung:



Meist ist es einfacher, Schaltungen, die den Booleschen Ausdrücken entsprechen, vom Ausgang her zu zeichnen – es wird auf diese Art übersichtlicher.

OR kann zwei unterschiedliche Bedeutungen annehmen. Als Booleschen Ausdruck kennen Sie es bisher in dieser Bedeutung:

Das eine OR das andere OR beide.

Es gibt aber eine zweite, für Logikschaltungen sehr wichtige andere Bedeutung dieses Ausdrucks:

Das eine OR das andere, aber nicht beide.

Ein Beispiel für das einschließende OR wäre, daß Sie zur Teilnahme an einem Zweirad-Rennen ein Fahrrad OR ein Motorrad haben müßten – Sie dürfen aber auch beides besitzen. Anders bei Gegenüberstellungen: Sie können nur entweder groß OR klein sein – da beides nicht gleichzeitig wahr sein kann, schließt dieses OR die Richtigkeit einer der beiden Behauptungen aus.

Das Exklusiv-OR

In Logikschaltungen wird das ausschließende OR (XOR) durch Kombination aus AND, OR und NOT-Gattern erzeugt.

Wahrheitstabelle:

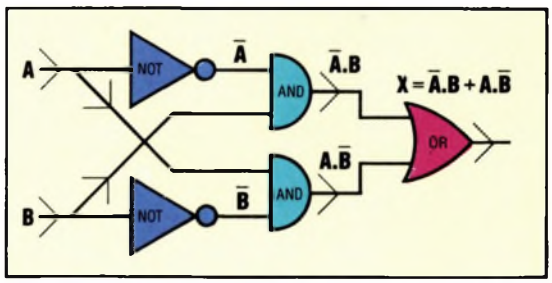
EINGÄNGE		AUSGANG
A	B	$A \veebar B$
0	0	0
0	1	1
1	0	1
1	1	0

Wie aus der zweiten und dritten Spalte sichtbar, erscheint am Ausgang die Eins, wenn

NOT(A) und B durch AND

OR

A und NOT(B) durch AND verknüpft werden. Der Boolesche Ausdruck dafür heißt $X=\overline{A}.B+A.\overline{B}$. Eine der möglichen Logikschaltungen für die ausschließende OR-Funktion (Exklusiv-OR) sieht so aus:



Für die Schaltung werden insgesamt fünf Gatter gebraucht. Sie werden später sehen, daß sich diese Anzahl durch Anwendung der Boo-

leschen Algebra auf vier vermindern läßt.

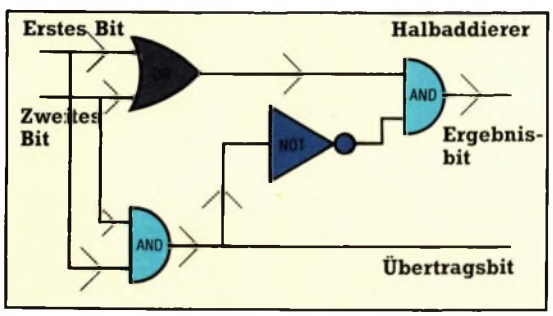
Obwohl die meisten Heimcomputer alle Rechenfunktionen beherrschen, wird nur die Addition von den üblichen Logik-Bausteinen direkt ausgeführt. Die anderen Rechenarten basieren auf der Verbindung integrierter Additions-Schaltungen mit spezieller Steuerungs-Software. Bevor wir näher auf die technische Realisierung der binären Addition eingehen, müssen wir einen Blick auf die Theorie des Additionsprozesses selbst werfen. Beispiel:

	8er	4er	2er	1er
	0	**1**	**0**	**1**
	0	**1**	**1**	**1**
	1	**1**	**0**	**0**
Übertrag	1	1	1	

Betrachten Sie die Zweier-Spalte allein, und untersuchen Sie die verschiedenen Ein- und Ausgaben von und zu dieser Spalte. Eingaben: Zwei Bits sollen addiert werden, außerdem kommt der Übertrag von der vorhergehenden Spalte hinzu. Ausgaben: ein Bit in die Zweier-Zeile des Ergebnisses, ein Bit Übertrag zur nächsten Spalte. Ein Addierer ist jedes Gerät, das diese Ein- und Ausgabeprozeduren fehlerlos vollführen kann. Volladdierer sind etwas komplizierter. Wir beginnen deshalb mit einem Halbaddierer, der den Übertrag von der vorhergehenden Spalte ignoriert. – Dadurch wird das Problem zweier Eingänge und zweier Ausgänge vorläufig umgangen. Die Wahrheitstabelle eines Halbaddierers sieht so aus:

EINGÄNGE		AUSGÄNGE	
Erstes Bit	**Zweites Bit**	**Übertragsbit**	**Ergebnisbit**
0	**0**	**0**	**0**
0	**1**	**0**	**1**
1	**0**	**0**	**1**
1	**1**	**1**	**0**

Es ist leicht ersichtlich, daß der Übertrag genau dann Eins wird, wenn das erste Bit AND das zweite Bit Eins sind. Das Ergebnisbit erhält man durch Verknüpfung beider Eingänge mit OR. Das Ergebnisbit ist also Eins, wenn das erste OR das zweite Bit Eins ist, aber NOT Eins, wenn das Übertragsbit Eins ist. Diese Schaltung erzeugt das Ergebnis:



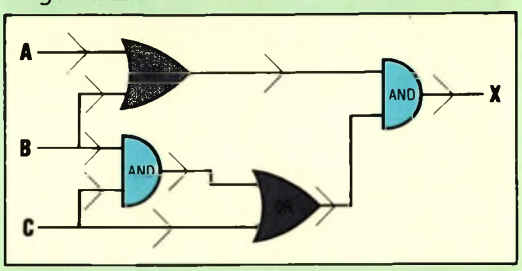
Übung 2

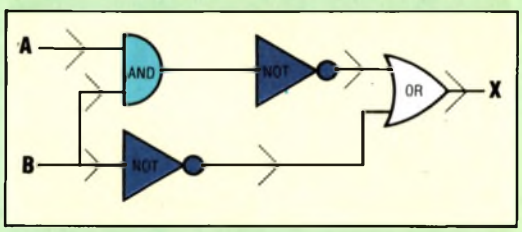
1) Zeichnen Sie zu den folgenden Booleschen Ausdrücken die entsprechende Logikschaltung:

a) X=(A+B).C
b) X=A.B+(A+C)
c) X=$\overline{A}$.B+(A+B)
d) X=$\overline{A}$.$\overline{B}$.(A+B)

2) Schreiben Sie den Booleschen Ausdruck für einen Halbaddierer. Verwenden Sie A und B als Eingänge.

3) Schreiben Sie den Booleschen Ausdruck für die beiden abgebildeten Schaltungen auf.





Lösungen zu Übung 1

1)

Hat Führerschein	Fahrlehrer dabei	Darf fahren
Falsch	Falsch	Falsch
Falsch	Wahr	Wahr
Wahr	Falsch	Wahr
Wahr	Wahr	Wahr

2)

Cassettenrecorder vorhanden	Diskettenlaufwerk vorhanden	Programm für anderen Computer	Programm läuft
Falsch	Falsch	Falsch	Falsch
Falsch	Falsch	Wahr	Falsch
Falsch	Wahr	Falsch	Wahr
Falsch	Wahr	Wahr	Falsch
Wahr	Falsch	Falsch	Wahr
Wahr	Falsch	Wahr	Falsch
Wahr	Wahr	Falsch	Wahr
Wahr	Wahr	Wahr	Falsch

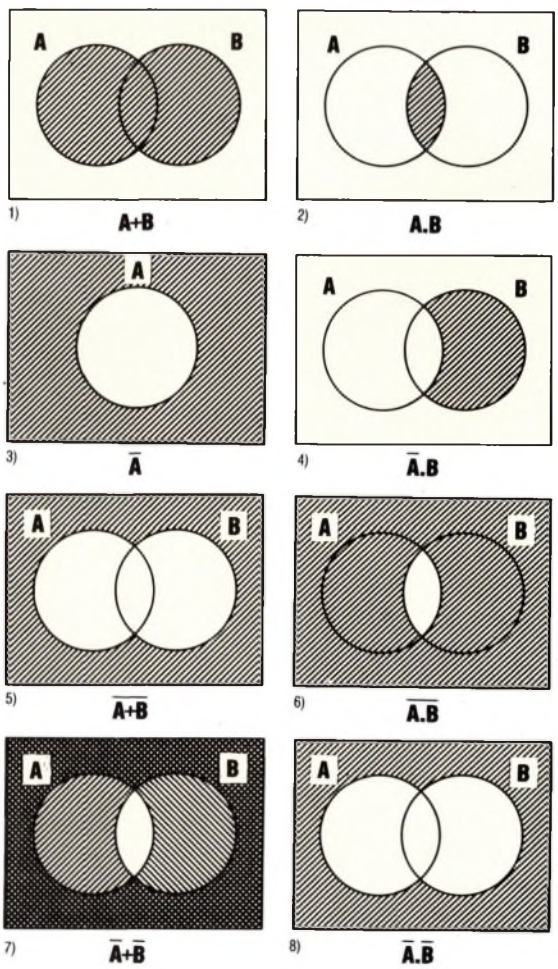
3)

A	B	P	Q	C
0	**0**	**1**	**0**	**0**
0	**1**	**1**	**1**	**1**
1	**0**	**0**	**1**	**0**
1	**1**	**0**	**1**	**0**

De Morgans Gesetz

Die Vereinfachung Boolescher Algebra führt zu logischen Ausdrücken, die mit einer kleinen Zahl von Operationen (AND, OR und NOT) auskommen. Durch klar definierte Regeln können Gatterbauteile eingespart werden – die gesamte Schaltung wird kompakter und ist preiswerter herzustellen.

Die Ausdrücke der Booleschen Algebra lassen sich grafisch sehr anschaulich durch Schnittmengen-Diagramme darstellen. Die Fläche innerhalb eines Rechteckes symbolisiert dabei alle Kombinationen der Wahrheitswerte, die eingegeben werden können. Die Kreise im Rechteck stehen für bestimmte Kombinationen von Eingabewerten. Hier sehen Sie einige Schnittmengen-Diagramme:



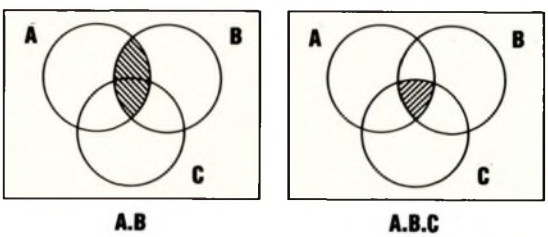
Beim Vergleich von Bild 5 mit Bild 7 sehen Sie, daß NOT (A OR B) etwas anderes ist als NOT (A) OR NOT (B). Ähnlich verdeutlichen Bild 6 und 8 den Unterschied zwischen NOT (A AND B) und NOT (A) AND NOT (B).

Um sich die AND- bzw. OR-Funktion als Diagramm vorzustellen, können Sie sich A.B als das Gebiet denken, wo die Fläche A die Fläche B überlappt – A + B ist dagegen die Vereinigung der Flächen von A und B. In der Booleschen Algebra gibt es viele Ausdrücke, die beim Anblick der dazugehörigen Grafik leichter zu verstehen sind und deren Richtigkeit sich optisch sofort erfassen läßt: Zeichnen Sie die Diagramme folgender Beziehungen: (0 steht dabei für „leere Menge")

1) $A.A = A$
2) $A.\overline{A} = 0$
3) $A.0 = 0$
4) $A.1 = A$
5) $A.(A + B) = A$
6) $A.(\overline{A} + B) = A.B$

Boolesche Gesetze

Die Regel von der Dualität ist bei der Vereinfachung Boolescher Ausdrücke eine große Hilfe. Sie beruht auf der Symmetrie der Funktionen AND und OR. Um das Gegenstück einer wahren Funktion zu finden, werden alle AND durch OR, alle Nullen durch Einsen ersetzt (natürlich auch umgekehrt). Aus dem fünften Ausdruck der obigen Liste würde so A + A.B=A. Da auch dieser Ausdruck richtig ist, zeigt das Schnittmengen-Diagramm: A.B wird vollständig von A umschlossen. Diese Methode läßt sich auch in Fällen mit drei Variablen anwenden, etwa bei A.B + A.B.C=A.B. Die beiden Diagramme bestätigen die Richtigkeit:



Versuchen Sie, auch die Gegenstücke der anderen fünf Funktionen zu finden, und kontrollieren sie die Gültigkeit mit entsprechenden Schnittmengen-Diagrammen.

Bei Betrachtung der Bilder 5 und 8 aus der ersten Diagramm-Reihe sehen Sie eine Beziehung, die immer wahr ist: $\overline{A+B}=\overline{A}.\overline{B}$. Bild 6 und 7 zeigen: $\overline{A.B}=\overline{A}+\overline{B}$. Diese beiden als „Gesetz von de Morgan" bezeichneten Beziehungen gelten auch bei mehr als zwei Variablen ($\overline{A+B+C} = \overline{A}.\overline{B}.\overline{C}$ und $\overline{A.B.C} = \overline{A}+\overline{B}+\overline{C}$). Dabei

wird in mehreren Schritten verfahren:

$(\overline{A+B}).\overline{C}$
$= \overline{A}.\overline{B}.\overline{C}$ (de Morgans Gesetz auf den Klammerausdruck angewandt)
$= \overline{A+B+C}$ (mit de Morgans Gesetz neu zusammengestellt)

Entsprechend der normalen Algebra gibt es auch in der Booleschen Algebra drei weitere gültige Gesetze. Das „Assoziativgesetz" erlaubt die Verschiebung von Klammern:

$(A.B).C=A.(B.C) = A.B.C$
$(A+B)+C = A+(B+C) = A+B+C$

Die Reihenfolge der Buchstaben kann mit dem „Kommutativgesetz" verändert werden:

$A.B = B.A$
$A+B = B+A$

Das „Distributivgesetz" erlaubt das Ausmultiplizieren von Klammern:

$A.(B+C) = A.B + A.C$

Einige Beispiele für Vereinfachungen mit Hilfe der genannten Gesetze:

1) Vereinfache $(\overline{A+B} + \overline{A}.B).B$
$= (\overline{A}.\overline{B} + \overline{A}.B).B$ (de Morgan)
$= \overline{A}.\overline{B}.B + \overline{A}.B.B.$ (Distributivgesetz)
$= 0 + \overline{A}.B$ ($\overline{B}.B = 0$, B.B=B)
$= \overline{A}.B$

2) Vereinfache $\overline{A}.\overline{B} + \overline{A}.B + A.B$
$= \overline{A}.(\overline{B}+B) + A.B$ (Distributivgesetz)
$= \overline{A} + A.B$ ($\overline{B} + B = 1$)
$= \overline{A} + B$ (Gegenstück der 6. Beziehung)

3) Vereinfache $\overline{\overline{A} + B} + \overline{\overline{A} + B} + \overline{A}.B$
$= \overline{\overline{A}}.\overline{B} + \overline{\overline{A}}.\overline{B} +$
$\overline{A}.B$ (de Morgan)
$= A.\overline{B} + A.B +$
$\overline{A}.B$ ($\overline{\overline{A}} = A$)
$= A.(\overline{B} + B) + \overline{A}.B$ (Distributivgesetz)
$= A + \overline{A}.B$ ($\overline{B} + B = 1$)
$= A + B$ (Gegenstück der 6. Beziehung)

Im letzten Abschnitt des Kurses haben wir die Schaltung des Exklusiv-OR-Gatters kennengelernt. Nun können Sie auch die Schaltung des Gatters ohne Beeinträchtigung der Funktion vereinfachen. Die Wahrheitstabelle des Gatters sieht so aus:

Eingänge		Ausgang
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Aus der Tabelle haben wir bereits ermittelt, daß $C=\overline{A}.B + A.\overline{B}$ ist. Durch eine kleine Vereinfachung würden fünf Logik-Gatter für diesen Ausdruck genügen. – Es gibt jedoch noch eine zweite Möglichkeit: C kann dann 1 sein, wenn A und B beide nicht 1 oder beide nicht 0 sind. C läßt sich also auch so ausdrücken:

$C = \overline{A.B + \overline{A}.\overline{B}}$

Dies läßt sich mit de Morgans Gesetz zu

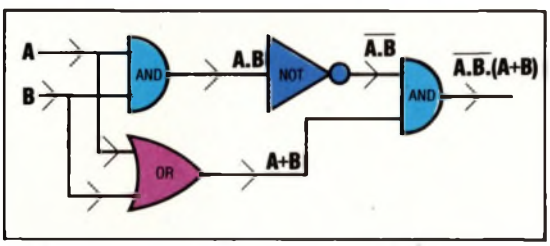
$C = (\overline{A.B}).(\overline{\overline{A}.\overline{B}})$

und in einem weiteren Schritt zu

$C=\overline{A.B}.(A + B)$

vereinfachen.

Dafür aber reichen vier Gatter:

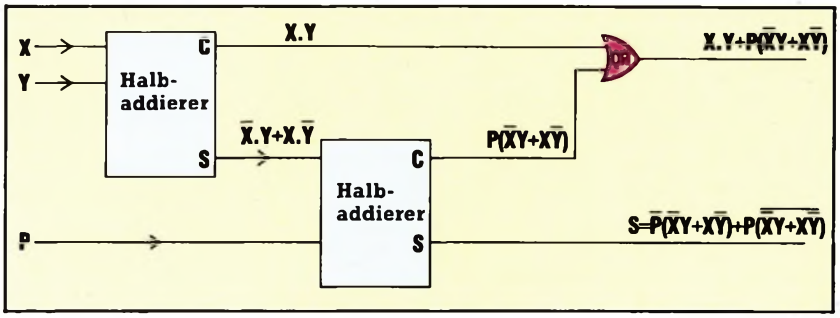


Einen einfachen Addierer zum Zusammenzählen von zwei Bits mit zwei Ausgängen für die Summe und den Übertrag haben Sie schon konstruiert. Dieser „Halbaddierer" hatte mit „X" und „Y" bezeichnete Eingänge. Das Verhalten des Ausgangs S ließ sich mit dem Booleschen Ausdruck $S = \overline{X}.Y + X.\overline{Y}$. beschreiben. De Morgans Gesetz kann auch diesen Ausdruck vereinfachen, nämlich zu $S=\overline{X.Y}.(X + Y)$. Der Übertrags-Ausgang ist einfach C =X.Y.

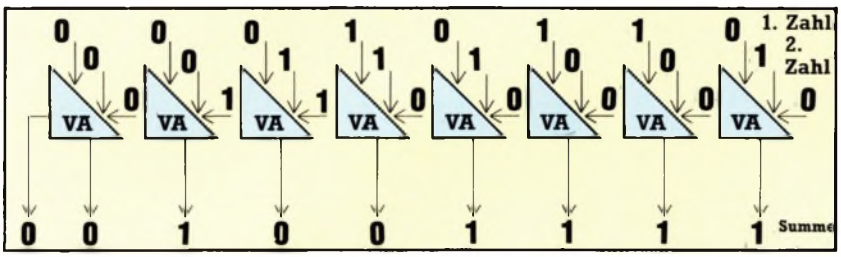
Bau eines Volladdierers

In jeder binären Rechnung müssen pro Spalte des Ergebnisses insgesamt drei Ziffern addiert werden: zwei Summanden und der Übertrag der vorhergehenden Spalte. Um diesen Vorgang zu simulieren, brauchen wir eine Schaltung mit drei Eingängen und zwei Ausgängen. Wird der Übertrag aus der vorhergehenden Zeile mit P bezeichnet, ergibt sich für den Volladdierer diese Wertetafel:

Eingänge			Ausgänge	
P	X	Y	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Hier die Schaltung eines Addierers für Acht-Bit-Zahlen, der aus acht Volladdierern aufgebaut ist:



Für die Fälle, in denen S = 1 ist, ergibt sich aus der Tabelle für S der Ausdruck:

$$S=\bar{P}.\bar{X}.Y + \bar{P}.X.\bar{Y} + P.\bar{X}.\bar{Y} + P.X.Y$$

Nach den neuen Regeln vereinfacht wird daraus

$$S=\bar{P}.(\bar{X}.Y + X.\bar{Y}) + P.(\bar{X}.\bar{Y} + X.Y)$$

(Distributivgesetz)

$$S=\bar{P}.(\bar{X}.Y + X.\bar{Y}) + P.\overline{(\bar{X}.Y + X.\bar{Y})}$$

(de Morgan)

Ähnlich kann aus der Wertetafel ein Ausdruck für C ermittelt werden:

$$C=\bar{P}.X.Y + P.\bar{X}.Y + P.X.\bar{Y} + P.X.Y$$

$$C=X.Y.(\bar{P} + P) + P.(\bar{X}.Y + X.\bar{Y})$$

(Distributivgesetz)

$$C=X.Y + P.(\bar{X}.Y + X.\bar{Y})$$

$(\bar{P} + P = 1)$

Beachten Sie, daß $\bar{X}.Y + X.\bar{Y}$ der Summenausgang eines Halbaddierers ist. Ein Volladdierer läßt sich also aus zwei Halbaddierern aufbauen.

ÜBUNG 3

1) Vereinfachen Sie die folgenden Ausdrücke:
   a) $A.(\bar{A}+\bar{B})$
   b) $X + Y.(X+Y) + X.(\bar{X} + Y)$
   c) $P.Q + \bar{P}.Q + \bar{P}.\bar{Q}$
   d) $\overline{\bar{X} + \bar{Y}.Z} + \bar{Z}.Y$

2) Eine Auto-Alarmanlage hat einen Ein/Ausschalter sowie Schalter an den vorderen Türen. Bei eingeschalteter Anlage und einer oder zwei offenen Türen gibt die Anlage Alarm. Zeichnen Sie eine Wahrheitstabelle mit drei Eingängen (Tür A, Tür B, Ein/Ausschalter) und einem (Alarm)-Ausgang. Schreiben Sie den Booleschen Ausdruck für die Funktion auf und zeichnen Sie die Logikschaltung dazu.

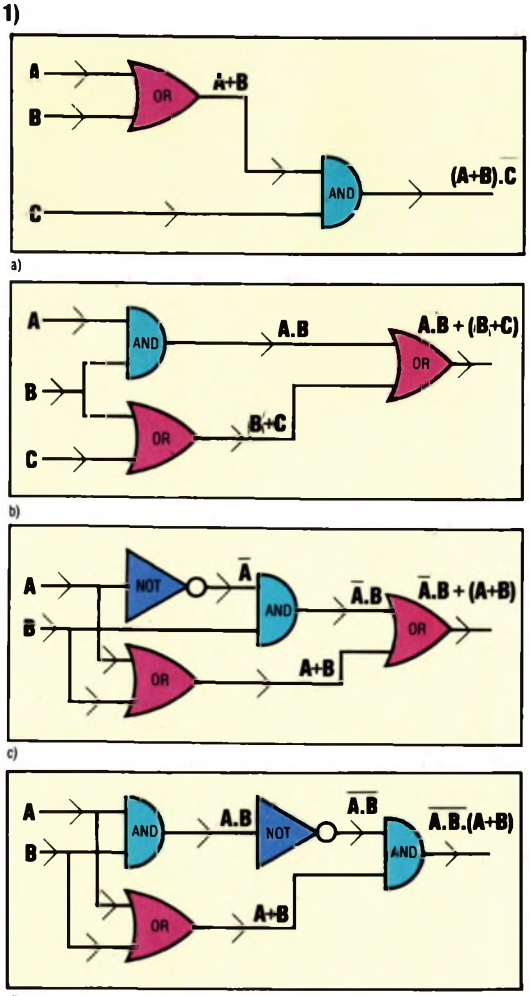
3) Eine Lampe im Treppenhaus soll mit drei Schaltern bedient werden, die an der Eingangstür sowie unten und oben an der Treppe liegen. Wie sieht die Logikschaltung aus?

4) Sie sind zu dritt auf einer einsamen Insel. Einer Ihrer Leidensgenossen spricht immer die Wahrheit, während der andere immer lügt. Wie können Sie herausfinden, wer die Wahrheit spricht? Die Wahrheitstabelle hilft Ihnen bei der Bewertung der Antworten. Beginnen Sie mit der Frage: „Sagst du immer die Wahrheit?"

		Mögliche Anworten	
		Ja	Nein
Was tut der Angesprochene	Lügt	1	0
	sagt die Wahrheit	1	0

Lösungen zu Übung 2

1)



2a) $C = A.B$
b) $S = \bar{A}.\bar{B}(A + B)$
3a) $X = (A + B).(B.C + C)$
b) $X = \bar{A}.\bar{B} + \bar{B}$

Bedingungs-abfragen

Die Logikfunktionen AND und OR sind auch in der Software von Bedeutung: Die meisten BASIC-Dialekte und Maschinensprachen enthalten AND und OR als wichtige Teile ihres Befehlssatzes.

In Maschinensprache und auch im BASIC gibt es eine Vielzahl von Anwendungsmöglichkeiten für Logik-Befehle. Meist sollen zwei oder mehrere Behauptungen zueinander in Beziehung gesetzt werden. Welches Ergebnis würden Sie beim folgenden BASIC-Programm vorhersagen?

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 5
30 IF I = 3 AND J = 2 THEN PRINT I,J
40 NEXT J
50 NEXT I
60 END
```

Das Programm durchläuft die beiden verknüpften Schleifen, druckt die Werte für I und J aber nur aus, wenn I = 3 und J = 2 ist. Auf dem Bildschirm zeigt sich dieses Resultat:

```
3          2
```

OR kann ähnlich eingesetzt werden. Ändern Sie Zeile 30 folgendermaßen:

```
30 IF I=3 AND J=2 OR J=4 THEN PRINT I,J
```

Das neue Programm ergibt diese Werte:

```
1          4
2          4
3          2
3          4
4          4
5          4
```

Der Computer führt die AND-Funktion vor der OR-Funktion aus. I und J werden ausgegeben, falls I=3 und J=2 sind oder wenn J=4 ist. Die Rechenhierarchie kann durch den Gebrauch von Klammern geandert werden. Welches Ergebnis erscheint bei einer neuen Zeile 30?

```
30 IF I=3 AND (J=2 OR J=4) THEN PRINT I,J
```

Um die Rechnerfunktionen zu steuern, werden in vielen Heimcomputern spezielle Register benutzt. Jedes Bit eines solchen Registers kann dabei eine eigene Aufgabe haben. So hat etwa der Commodore 64 ein Acht-Bit-Register zum Ein- und Ausschalten der Sprites. Dabei ist ein Bit für jedes der acht Sprites zuständig. Wenn ein Bit des Registers auf Eins gesetzt wird, ist das dazugehörige Sprite auf dem Bildschirm sichtbar, während es bei einer Null ausgeschaltet bleibt. Mit BASIC-Befehlen kann durch POKEn der entsprechenden achtstelligen Binärzahl in das Register jede beliebige Sprite-Kombination verwirklicht werden. Allerdings berücksichtigt diese Methode nicht den Zustand des Registers vor dem POKE-Befehl – ein Sprite, das vorher eingeschaltet war, verschwindet möglicherweise vom Bildschirm. Die Lösung des Problems besteht darin, das gewünschte Bit zu isolieren und es zu verändern, ohne dabei die anderen Bits zu beeinflussen.

Um diese Technik einmal vorzuführen, nehmen wir an, daß die Sprites 0, 1, 5 und 6 eingeschaltet sind. Die Werte im Spriteregister sehen dann so aus:

Spritenummer	7	6	5	4	3	2	1	0
Ein/Aus-Bit	0	1	1	0	0	0	1	1

Sprite 4 wird nun eingeschaltet, indem mit PEEK der Registerinhalt abgerufen und durch OR mit 16 (binär 00010000) verknüpft und das Ergebnis wieder ins Register gePOKEt wird.

Ursprüngliches Byte	0	1	1	0	0	0	1	1
Verknüpft (OR) mit	0	0	0	1	0	0	0	0
Ergibt neues Byte	0	1	1	1	0	0	1	1

Mit dem BASIC-Befehl POKE reg, PEEK(reg)OR 16 läßt sich Bit 4 im Register einschalten. Zum Ausschalten wird der Registerinhalt mit PEEK abgerufen und durch AND mit 239 verknüpft:

Neues Byte	0	1	1	1	0	0	1	1
AND	1	1	1	0	1	1	1	1
Ursprüngliches Byte	0	1	1	0	0	0	1	1

Das Ergebnis (239) läßt sich durch Subtrahieren der 16 von der Zahl 255 berechnen. Mit POKE reg, PEEK(reg)AND239 ist das Register jetzt wieder im alten Zustand.

Besonders häufig kommt die beschriebene Technik in Maschinenspracheprogrammen vor – die Veränderung von Kontrollregistern stellt oft den Kernpunkt solcher Programme dar.

Lösungen zu Übung 3

1a) $A.(\bar{A}+B)$
$=A.\bar{A}+A.B.$ (Distributivgesetz)
$=A.B$ ($A.\bar{A}=0$)

b) $X+Y.(X+Y)+X.(\bar{X}+Y)$
$=X+Y+X.(\bar{X}+Y)$ (Beziehung 5)
$=X+Y+X.Y$ (Beziehung 6)
$=X+Y$ (Absorption)

c) $P.Q+\bar{P}.Q+\bar{P}.\bar{Q}$
$=P.Q+\bar{P}.(Q+\bar{Q})$ (Distributivgesetz)
$=P.Q+\bar{P}$ ($Q+\bar{Q}=1$)
$=\bar{P}+Q$ (Gegenstück zur Beziehung 6)

d) $\bar{X}.\bar{Y}.Z+\bar{Z}.Y$
$=\bar{X}.Y.Z.Z.Y$ (de Morgan)
$=X.Y.Z.(Z+\bar{Y})$ ($\bar{X}=X$, de Morgan)
$=X.Y.Z.Z+X.Y.Z.\bar{Y}$ (Distributivgesetz)
$=X.Y.Z+0$ ($Z.Z=Z$, $Y.\bar{Y}=0$)
$=X.Y.Z$

3) Wenn die drei Schalter mit X,Y und Z sowie die Lampe als P bezeichnet werden, ergibt sich diese Wertetabelle:

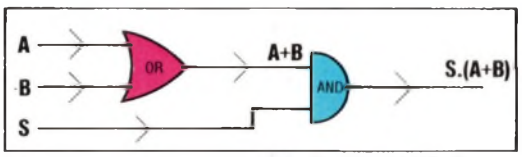
Eingänge			Ausgang
X	Y	Z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$P=\bar{X}.\bar{Y}.Z+X.Y.\bar{Z}+X.\bar{Y}.\bar{Z}+X.Y.Z$
$=Z.(\bar{X}.\bar{Y}+X.Y)+Z.(\bar{X}.Y+X.\bar{Y})$ (Distributivgesetz)
$=Z.(\overline{\bar{X}.\bar{Y}+X.Y})+Z.(\bar{X}.Y+X.\bar{Y})$ (de Morgan)

2) Wertetabelle der Alarmanlage:

Eingänge			Ausgang
A	B	S	Alarm
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

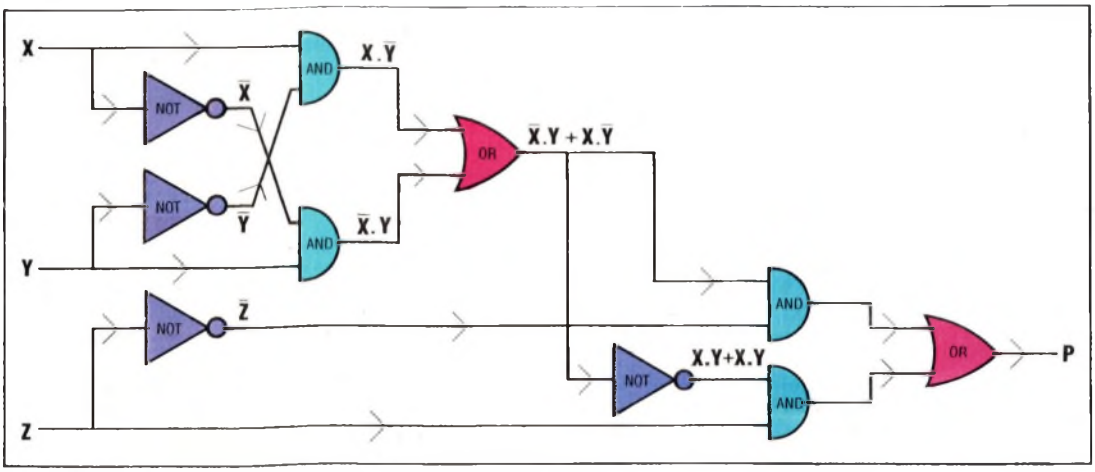
Alarm $=\bar{A}.B.S+A.\bar{B}.S+A.B.S$
$=\bar{A}.B.S+A.S.(\bar{B}+B)$ (Distributivgesetz)
$=\bar{A}.B.S+A.S$ ($\bar{B}+B=1$)
$=S.(A+\bar{A}.B)$ (Distributivgesetz)
$=S.(A+B)$ (Gegenstück zur Beziehung 6)



4) Die Wahrheitstafel zeigt, daß man mit der Frage „Sagst Du die Wahrheit?" nicht weit kommt – in beiden Fällen würde die Antwort gleich lauten. Die Tabelle entspricht in ihrer Form der Funktion $X.Y+\bar{X}.Y$, verkürzt als Y. Das bedeutet, die Antwort hängt nur von einer Variablen ab und kann deshalb nicht dabei helfen, zwischen den Personen zu unterscheiden. Bei der Frage „Haben Schweine Flügel?" sieht die Tabelle zum Glück schon anders aus:

		Mögliche Antwort	
		JA	NEIN
Was tut der Gefragte möglicherweise	lügt	1	0
	sagt die Wahrheit	0	1

Damit haben wir die Wahrheitstabelle der Funktion $X.\bar{Y}+\bar{X}.Y$, die der Exklusiv-OR-Funktion entspricht – damit läßt sich der Lügner ohne weiteres identifizieren.

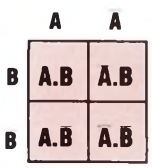


Karnaugh-Tafeln

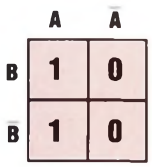
Karnaugh-Tafeln sind bei der Vereinfachung von Logikschaltungen unentbehrlich. Dabei ersetzen sie nicht die bereits behandelten rechnerischen Methoden, können aber eine große Entlastung sein, wenn schwierige Ausdrücke der Booleschen Algebra mit mehreren Variablen vereinfacht werden sollen.

Karnaugh-Tafeln (auch als k-Tafeln bezeichnet) sind eine Erweiterung der Venn-Diagramme, die schon aus früheren Kursabschnitten bekannt sind. Sie dienen zur bildlichen und vereinfachten Darstellung logischer Ausdrücke. Die Form einer solchen Tafel wird durch die Menge der Buchstaben oder Variablen der Funktion bestimmt. Besonders gut läßt sich mit k-Tafeln für zwei, drei oder vier Variablen arbeiten.

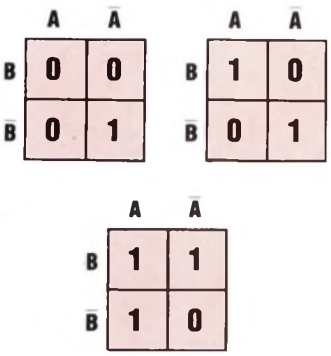
Zwei Variablen: Jede Zelle einer k-Tafel für zwei Variablen entspricht einer AND-Funktion, wie sie hier im Bild gezeigt wird:



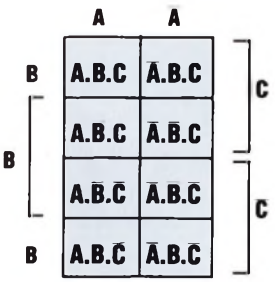
Um den Ausdruck $AB + A\bar{B}$ als k-Tafel darzustellen, werden Einsen an den entsprechenden Stellen eingetragen:



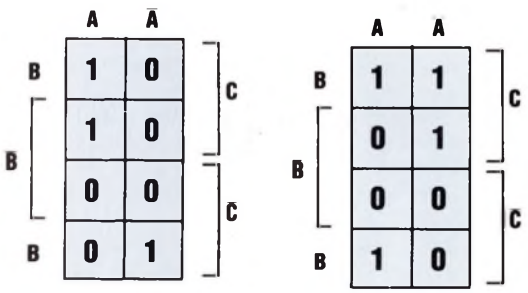
Hier drei weitere Beispiele, die entsprechend die Funktionen $\bar{A}\bar{B}$, $AB+\bar{A}\bar{B}$ und $AB + \bar{A}B + A\bar{B}$ darstellen:



Drei Variablen: In diesem Fall nimmt die Anzahl der Zellen um den Faktor Zwei zu ($2^3 = 8$ Zellen). Die Basis-k-Tafel für 3 Variablen:



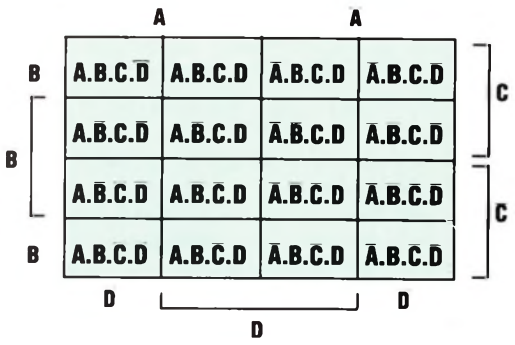
Diese k-Tafeln verkörpern die Ausdrücke $AC + \bar{A}B\bar{C}$ und $AB + \bar{A}C$:



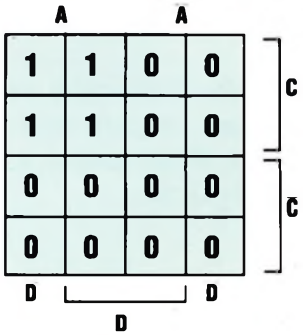
$ABC + A\bar{B}C + \bar{A}B\bar{C}$
$= AC(B + \bar{B}) + \bar{A}B\bar{C}$
$= AC + \bar{A}B\bar{C}$
$ABC + AB\bar{C} + \bar{A}BC + \bar{A}B\bar{C}$
$= AB(C + \bar{C}) + \bar{A}C(B + \bar{B})$
$= AB + \bar{A}C$

Beachten Sie, daß beide Ausdrücke mit dem Booleschen Gesetz der Universalität oder Identität vereinfacht wurden – ein A, das durch OR mit seiner Negation ($\bar{A}$) verknüpft wird, ergibt die 1.

Vier Variablen: Bei vier Variablen wird die Tafel bereits recht kompliziert (sie hat $2^4 = 16$ Zellen). Durch das unveränderte Prinzip ist sie aber leicht zu interpretieren:

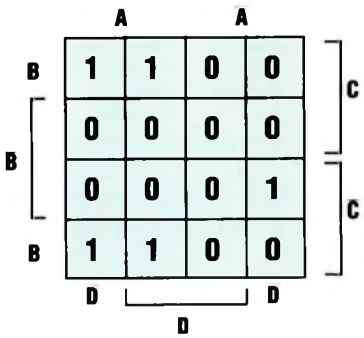


Hier sehen Sie eine k-Tafel mit den dazugehörigen Vereinfachungen:



$AB\overline{C}\overline{D} + ABCD + A\overline{B}\overline{C}\overline{D} + A\overline{B}CD$
$= ABC(\overline{D} + D) + A\overline{B}C(\overline{D} + D)$
$= ABC + A\overline{B}C$
$= AC(B + \overline{B})$
$= AC$

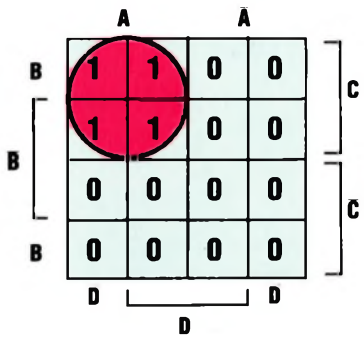
Ein weiteres Beispiel:

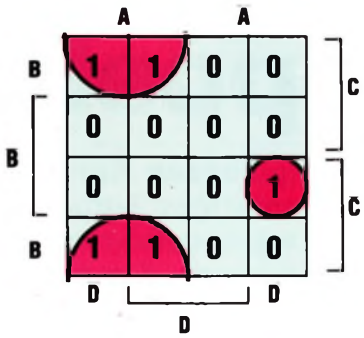


$AB\overline{C}\overline{D} + ABCD + \overline{A}\overline{B}\overline{C}\overline{D} + AB\overline{C}\overline{D} + AB\overline{C}D$
$= ABC(\overline{D} + D) + \overline{A}\overline{B}\overline{C}\overline{D} + AB\overline{C}(\overline{D} + D)$
$= ABC + \overline{A}\overline{B}\overline{C}\overline{D} + AB\overline{C}$
$= AB(C + \overline{C}) + \overline{A}\overline{B}\overline{C}\overline{D}$
$= AB + \overline{A}\overline{B}\overline{C}\overline{D}$

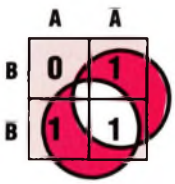
Stellung der Einsen ist wichtig

Wenn Sie die Stellung der Einsen in einer Tafel betrachten, werden Sie ein Muster erkennen. So haben im ersten Beispiel alle Ausdrücke mit AC eine Eins in der entsprechenden Zelle, im zweiten Beispiel alle Ausdrücke mit AB. Das führt zu einem Verfahren, mit dem sich Boolesche Ausdrücke leichter vereinfachen lassen. Hier ein Beispiel:



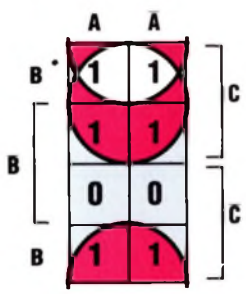


Mit ein wenig Übung können Sie Gruppen von Einsen herausgreifen und damit einfachere Ausdrücke formulieren. Hier etwa $\overline{A}B + A\overline{B} + \overline{A}\overline{B}$.



Aus einer k-Tafel mit zwei Variablen können zwei Gruppen von Einsen herausgelöst werden. Die eine Gruppe umfaßt alle NOT(B)-Fälle, die andere alle NOT(A)-Fälle, der Ausdruck läßt sich also zu $\overline{A} + \overline{B}$ vereinfachen. Mit dem Gesetz von de Morgan wird daraus $\overline{A.B}$. Ist dieses Ergebnis durch eine k-Tafel schneller zu erzielen?

Hier noch ein etwas schwierigeres Beispiel mit drei Variablen:
$ABC + A\overline{B}C + \overline{A}BC + AB\overline{C} \cdot \overline{A}B\overline{C} + \overline{A}\overline{B}C$



Die Gruppe der vier Einsen oben in der Tafel stellt alle Möglichkeiten dar, in denen C wahr ist. Die unteren und oberen Zeilen stehen für die Fälle, in denen B wahr ist. Der vereinfachte Ausdruck lautet also: B + C.

Im nächsten Kursabschnitt werden wir diesen Einstieg in die Anwendung von Karnaugh-Tafeln weiter vertiefen und dabei Ausdrücke mit vier Variablen untersuchen. Auch die Aufgabe der k-Tafeln beim Schaltungsentwurf soll gezeigt werden – dazu sind alle in diesem Kurs behandelten Theorien nötig.

Übung 4
Zeichnen Sie eine k-Tafel für drei Variablen, um die folgenden Ausdrücke zu vereinfachen:

a) $\overline{A}.B.C + \overline{A}.\overline{B}.C + \overline{B}.\overline{C} + \overline{A}.B.\overline{C}$
b) $A.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.B.\overline{C}$

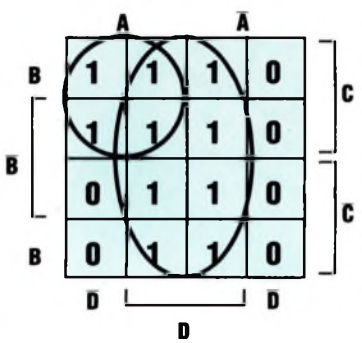
Schaltpläne

Den Nutzen von Karnaugh-Tafeln bei der Vereinfachung Boolescher Ausdrücke mit zwei oder drei Variablen kennen Sie bereits. Jetzt sind die schwierigen Fälle mit vier Variablen an der Reihe. Außerdem werden Sie mit der Anwendung von Karnaugh-Tafeln im Schaltungsentwurf vertraut gemacht.

Vier Variablen: Eine Karnaugh-Tafel für eine Booleschen Ausdruck mit vier Variablen wirkt auf den ersten Blick etwas kompliziert. Aber keine Angst, mit den schon bekannten Regeln läßt sich auch eine solche Tafel problemlos bearbeiten. Nehmen Sie einmal an, daß Sie diesen Ausdruck vereinfachen sollen:

$$ABC\overline{D} + ABCD + \overline{A}BCD + A\overline{B}\overline{C}\overline{D} + \overline{B}CD + A\overline{B}CD + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}D + B\overline{C}D$$

Dafür brauchen Sie eine k-Tafel für vier Variablen. Weil der Ausdruck aber aus acht Elementen besteht, müssen insgesamt zehn Einsen in die Tafel eingetragen werden (sowohl $\overline{B}CD$ als auch $B\overline{C}D$ stellen je zwei Fälle dar). Hier die k-Tafel:



Die k-Tafel zeigt eine zentrale Gruppe von acht Einsen, die für jede mögliche Kombination mit D stehen. Eine zweite Gruppe von vier Einsen in der linken oberen Ecke stellt alle Fälle mit A AND C dar. Der Ausdruck läßt sich also zu A AND C OR D (A.C + D) vereinfachen.

Ausdruck verändern

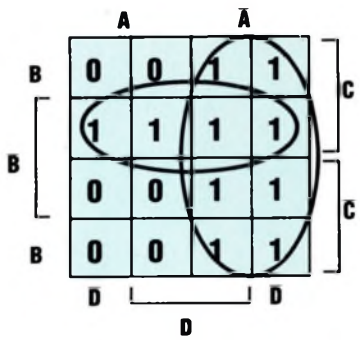
Gelegentlich muß ein Ausdruck verändert werden, damit er sich als k-Tafel schreiben läßt. Hier ein Beispiel:

$$A+B+\overline{C}+\overline{A}.B+\overline{B+\overline{C}}$$

Die entsprechende Tafel kann nur nach Anwendung des Gesetzes von de Morgan erstellt werden. Der Ausdruck heißt dann:

$$\overline{A}.\overline{B}.\overline{C} + \overline{A}.B + \overline{B}.C$$

Und das ist die zugehörige Karnaugh-Tafel:



Mit der Tafel finden Sie die Vereinfachung zu $\overline{A} + \overline{B}.C$. Ein zweites Mal mit de Morgans Gesetz behandelt, ergibt sich schließlich dieser Ausdruck:

$$\overline{A\,(B+\overline{C})}$$

Beispiel 1: Dreißig Tage

Sie kennen sicher den Trick, durch Abzählen an den Fingerknöcheln zu bestimmen, ob ein Monat dreißig oder einunddreißig Tage hat. Aber wie sieht eine elektronische Schaltung für diese Bestimmung aus? Dazu ordnen wir jedem Monat einen Vier-Bit-Code zu – von 0001 für den Januar bis 1100 für den Dezember. Nach Eingabe dieses Codes soll die Schaltung für Monate mit dreißig Tagen eine 1 ausgeben.

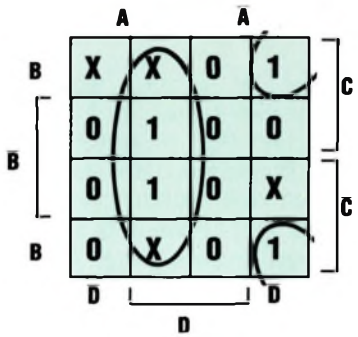
Die Wertetabelle einer solchen Schaltung:

Monat	Eingaben				Ausgabe
	A	B	C	D	S
	0	0	0	0	X
JAN	0	0	0	1	0
FEB	0	0	1	0	0
MÄR	0	0	1	1	0
APR	0	1	0	0	1
MAI	0	1	0	1	0
JUN	0	1	1	0	1
JUL	0	1	1	1	0
AUG	1	0	0	0	0
SEP	1	0	0	1	1
OKT	1	0	1	0	0
NOV	1	0	1	1	1
DEZ	1	1	0	0	0
	1	1	0	1	X
	1	1	1	0	X
	1	1	1	1	X

Das Ergebnis X ist in der Wertetabelle für ungültige Eingaben vorgesehen, wir gehen aber davon aus, das solche Eingaben nicht erfolgen. Für alle Fälle, in denen S=1 ist, läßt sich aus der Tabelle dieser Ausdruck formen:

$$S = \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.C.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.D$$

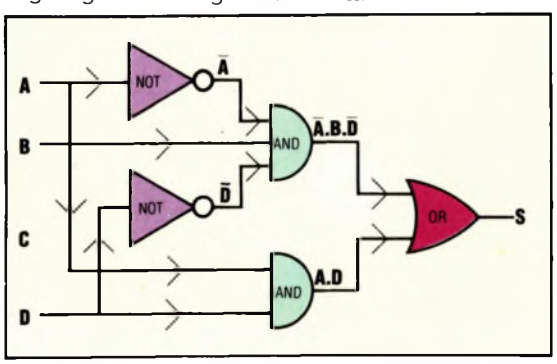
Zusammen mit den ungültigen Eingaben (X) ergibt sich die k-Tafel:



Die Tafel hilft beim Vereinfachen des Ausdrucks auf

$$A.D + \overline{A}.B.\overline{D}$$

Danach läßt sich die Schaltung für eine „Dreißig-Tage-Meldung" entwerfen:



Beispiel 2: Ungerade Zahlen

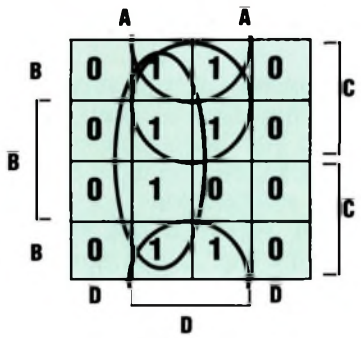
Dezimal-zahl	Eingaben A	B	C	D	Ausgabe S
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Die Zahlen 0 bis 15 können auch durch vier Binärziffern (0000 bis 1111) dargestellt werden. Die geplante Schaltung soll bei einer im Binärcode eingegebenen ungeraden Zahl, die größer als zwei ist, eine 1 ausgeben.

Der erste Schritt ist das Aufstellen der Wertetafel für alle möglichen Fälle. Wieder muß für alle Fälle, in denen S wahr (=1) ist, der entsprechende Boolesche Ausdruck formuliert werden:

$$S = \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.B.\overline{C}.D + A.B.C.D$$

Daraus entsteht diese k-Tafel:



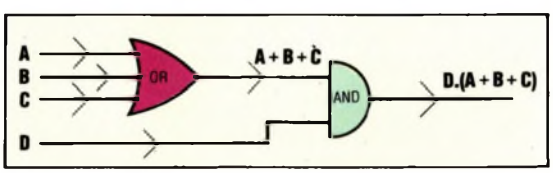
Aus der Tafel können Sie drei Vierergruppen isolieren. Dieser Ausdruck beschreibt sie:

$$S = A.D + C.D + B.D$$

Vereinfacht durch Anwendung des Distributiv-Gesetzes wird daraus:

$$S = D.(A + B + C)$$

Hier die dazugehörige Schaltung:



Der nächste Kursabschnitt wird Ihre Kenntnisse durch Wiederholung der wichtigsten Regeln und viele interessante Übungsaufgaben weiter vertiefen.

Übung 5

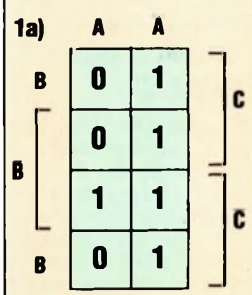
1) Vereinfachen Sie die mit Hilfe der Karnaugh-Tafeln folgende Boolesche Ausdrücke:

a) $A.B.C + A.\overline{B}.\overline{C} + \overline{A}.\overline{C} + \overline{A}.B.C + A.B.\overline{C}$
b) $\overline{B + C} + B.\overline{C} + A.C$
c) $A.\overline{B}.D + \overline{A}.D + A.B.C.D + A.B.\overline{C} + \overline{A}.B.\overline{C}.\overline{D}$

2) Entwerfen Sie eine Schaltung für die Eingabe der Binärdarstellung von Ziffern zwischen 0 und 7. Die Schaltung soll eine 1 ausgeben, wenn die Eingabe entweder eine ungerade Zahl oder ein Vielfaches von 3 ist (3 oder 6). Vereinfachen Sie die Logikfunktion mit der Wahrheitstabelle und den k-Tafeln.

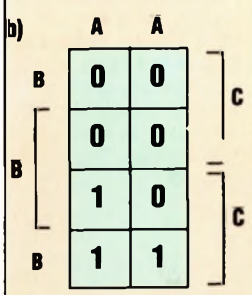
Lösungen der Übung 4

1a)



$\overline{A} + B.\overline{C}$

b)



$(A + B).\overline{C}$

Ist doch logisch!

In der Logik haben wir jetzt festen Boden unter den Füßen, speziell auf dem Gebiet der Booleschen Algebra und bei den logischen Grundfunktionen. Dieser Kursteil faßt das bisher Behandelte noch einmal zusammen und enthält darüber hinaus einige Wiederholungsaufgaben und die dazugehörigen Lösungen.

Lassen Sie uns kurz auf den Inhalt der früheren Kursabschnitte zurückblicken: In einem Computer gibt es zahlreiche für spezielle Zwecke entworfene Logikschaltungen. Dazu zählt der von uns gebaute Halbaddierer, der zusammen mit anderen Addierschaltungen die Berechnung einer Summe aus zwei Binärzahlen ermöglicht. Die Schaltung imitiert das Rechnen „per Hand" – bei einer Addition nimmt sie einen Übertrag auf die nächste Spalte vor.

Die Grundelemente, die sogenannten „logischen Gatter", kamen in der Schaltung zum Einsatz. Die Bezeichnung logisches Gatter korrespondiert mit ihrer Funktion (AND, OR und NOT). Durch Wahrheitstabellen konnten wir jedem Gatter die einer bestimmten Kombination von Eingabewerten entsprechenden Ausgaben zuordnen. Zwei Eingänge ergaben vier (2^2), drei Eingänge acht (2^3) Kombinationen usw. Es wurde ferner die Verknüpfung mehrerer Gatter zu einer Schaltung mit der gewünschten Entsprechung zwischen Ein- und Ausgabe gezeigt und durch Wahrheitstabellen beschrieben. Die Fünf-Gatter-Schaltung der ausschließenden OR-Funktion etwa gab nur dann eine 1 aus, wenn ein Eingang 1 war.

Mit speziellen Symbolen kann die Kombination logischer Funktionen wie beim gewohnten Rechnen auf dem Papier vorgenommen werden. Dieser Bereich der mathematischen Logik wird nach seinem Begründer George Boole (1815–64) „Boolesche Algebra" genannt. Jedes der drei logischen Grundelemente hat in der Booleschen Algebra sein besonderes Symbol:

A AND B	A.B
A OR B	A+B
NOT A	$\overline{A}$

Wie in der Zahlenmathematik gibt es auch in der Logik feste Regeln für den Umgang mit Buchstaben und Funktionen, die zur Vereinfachung logischer Ausdrücke genutzt werden können. Diese „Rechenvorschriften" haben wir hier in der rechten Spalte zu einer Tabelle zusammengefaßt.

Die Anwendung dieser Regeln hilft bei der Vereinfachung einer logischen Funktion und vermindert dadurch die Anzahl der zu ihrer Realisierung notwendigen Gatterbausteine. Dem gleichen Zweck dienen auch die bereits erwähnten Karnaugh-Tafeln. Diese ersetzen zwar nicht die algebraischen Methoden, können jedoch beim Versuch der Vereinfachung einer logischen Funktion eine große Arbeitserleichterung darstellen. Mit diesen Tafeln lassen sich innerhalb einer Wahrheitstabelle Gruppen von zwei, vier oder acht Ausdrücken bilden. Diese können zu einfacheren Ausdrükken zusammengefaßt werden. In der Praxis wird oft mit einer Kombination aus k-Tafeln und algebraischen Methoden gearbeitet.

Sonderfälle	
Beziehung	**Entsprechung**
$A.A = A$	$A+A = A$
$A.\overline{A} = 0$	$A+\overline{A} = 1$
$A.0 = 0$	$A+1 = 1$
$A.1 = A$	$A+0 = A$
$A.(A+B) = A$	$A+A.B = A$
$A.(\overline{A}+B) = A.B$	$A+\overline{A}.B = A+B$
Gesetze von de Morgan	
1) $\overline{A+B} = \overline{A}.\overline{B}$	
2) $\overline{A.B} = \overline{A}+\overline{B}$	
Assoziativgesetz	
$A.(B.C) = (A.B).C = A.B.C$	
$A+(B+C) = (A+B)+C = A+B+C$	
Kommutativgesetz	
$A.B = B.A$	
$A+B = B+A$	
Distributivgesetz	
$A.(B+C) = A.B+A.C$	

Auch den Nutzen der logischen AND- und OR-Funktion für die BASIC-Programmierung haben wir schon behandelt – sie können etwa zum Setzen einzelner Bits eines Registers hilfreich sein.

Als Schlußpunkt des ersten Kursabschnittes konnten wir durch die Kombination unserer Kenntnisse von k-Tafeln, Boolescher Algebra und logischen Gattern Schaltungen entwerfen, deren Ausgabe sich nach dem von uns festgelegten Zweck richtete. Wenn Sie sich auch beim Lösen der folgenden Wiederholungsaufgaben sicher fühlen, steht der nächste Schritt an: die fortgeschrittene Theorie der Logik.

Wiederholungsaufgaben

1) Einer Rockband wird die Produktion einer Hit-Single versprochen, wenn sie einen guten Video-Clip liefert UND die Plattenfirma Werbegeschenke zur Platte dazugibt ODER wenn die Gruppe im Fernsehen auftritt. Zeichnen Sie dazu eine Wahrheitstabelle. Wie stehen die Chancen für die Gruppe, einen Hit zu landen, wenn alle Möglichkeiten gleich wahrscheinlich sind?

2) Für den Schul-Schachklub sollen Mitgliedskarten ausgegeben werden. Die Mitgliedschaft soll beschränkt sein auf:

i) Bedienstete der Schule
ii) Schüler des vierten und fünften Semesters mit Leistungskurs Mathematik und Physik
iii) Schüler des sechsten Semesters mit Leistungskurs Mathematik oder Physik

Entwerfen Sie ein Gerät, das die Mitgliedskarten auf Knopfdruck ausgibt. Das Gerät hat folgende Wahltasten:

A= Schüler des vierten Semesters
B= Schüler des fünften Semesters
C= Schüler des sechsten Semesters
D= Schulbediensteter
E= Leistungskurs Mathematik
F= Leistungskurs Physik

Zeichnen Sie die Schaltung dazu.

3) Ein Computer steuert den Bildschirm über das Register 23148(dezimal). Bit 0 ist das niedrigste, Bit 7 das höchste Bit im Register. Der Bildschirm wird auf hohe Auflösung umgestellt, wenn Bit 4 und 5 auf 1 gesetzt sind. Dabei dürfen die anderen Bits nicht verändert werden. Schreiben Sie ein BASIC-Programm, das

a) auf hohe Auflösung umschaltet,
b) die hohe Auflösung wieder ausschaltet.

4) Ein Tresor-Schloß hat mehrere Schalter, die mit Schlüsseln bedient werden müssen. Der Filialleiter, sein Stellvertreter und der Hauptkassierer haben je einen Schlüssel. Mit zwei Schlüsseln kann der Tresor geöffnet werden, allerdings gibt es dabei Bedingungen:

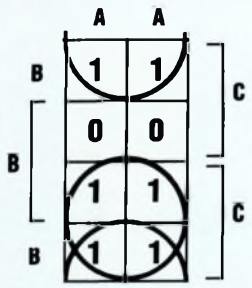
a) In der Hauptgeschäftszeit zwischen 9 und 17 Uhr kann der Tresor nur geöffnet werden, wenn der Hauptkassierer dabei ist.
b) Zu anderen Zeiten läßt sich der Tresor nur vom Filialleiter zusammen mit einer der beiden anderen Personen öffnen.

Entwerfen Sie die entsprechende Schaltung!

5) Um in einem digitalen Datenübertragungs-System besonders hohe Zuverlässigkeit zu gewährleisten, wird jedes Datenbit gleichzeitig in drei unabhängige Leitungen gespeist. Bei richtiger Übertragung liefern alle Leitungen beim Empfänger denselben Wert. Entwerfen Sie eine Schaltung, die bei falschen Daten einer Leitung deren Werte nach dem Mehrheitsprinzip korrigiert.

Lösungen der Übung 5

1a)



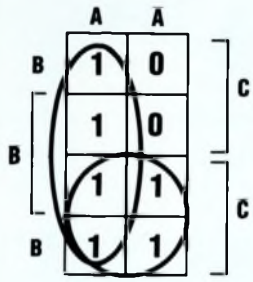
$= \overline{C} + B$

(alternativ: $= \overline{C.\overline{B}}$)

b)

$\overline{B + C} + B.\overline{C} + A.C$

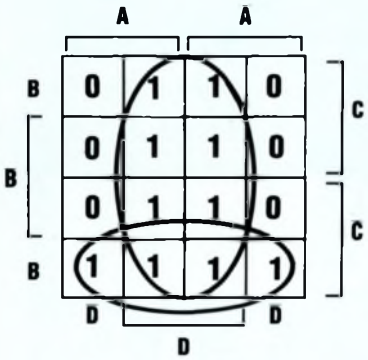
$= \overline{B}.\overline{C} + B.\overline{C} + A.C$ (Gesetz von de Morgan)



$= A + \overline{C}$

(alternativ: $= \overline{A}.C$)

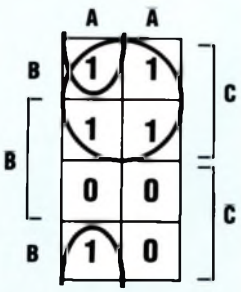
c)



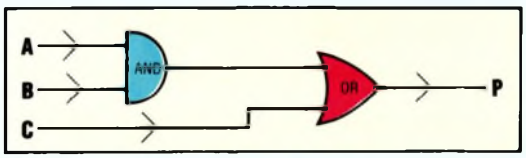
$= D + B.\overline{C}$

2) Die Wahrheitstabelle:

Dezimal	A	B	C	P
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



Aus der k-Tafel ergibt sich der Ausdruck P=C+A.B.. Hier die Schaltung dazu:



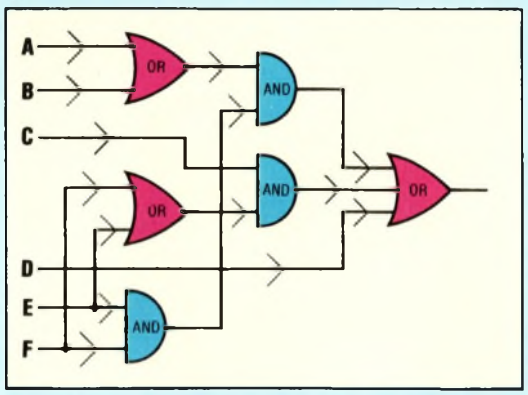
Lösungen der Wiederholungsaufgaben

1)

Guter Video-Clip	Werbe-geschenk	In der Hitliste	Single wird produziert
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Wahrscheinlichkeit des Erfolgs= 5/8 = 62,5%

2)



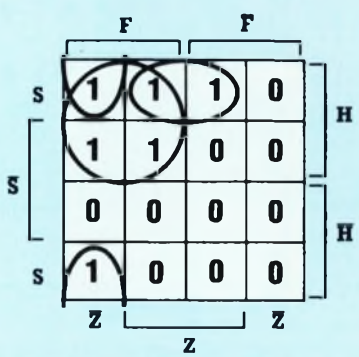
3a) Zum Einschalten der hohen Auflösung:
POKE23148,PEEK(23148)OR48

b) Zum Ausschalten der hohen Auflösung:
POKE23148,PEEK(23148)AND207

4) Die Wahrheitstabelle:

Filial-leiter	Stell-vertreter	Haupt-kassierer	Zeit v. 9–17 Uhr	Tresor geht auf
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

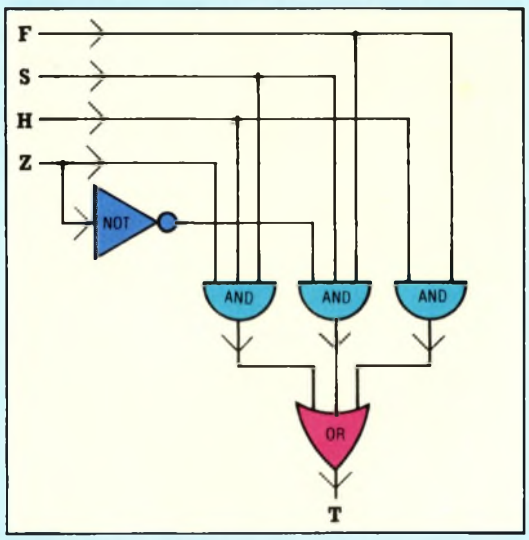
Die k-Tafel:



Hieraus ergibt sich der Ausdruck
T=F.H+F.S.Z+S.H.Z. – die Richtigkeit wird bei der Übertragung in Langform klarer, er besagt nämlich: Der Tresor läßt sich von

i) Filialleiter und Hauptkassierer jederzeit,
ii) Filialleiter und Stellvertreter außerhalb der Geschäftszeit und von
iii) Stellvertreter und Hauptkassierer in der Zeit von 9 bis 17 Uhr öffnen.

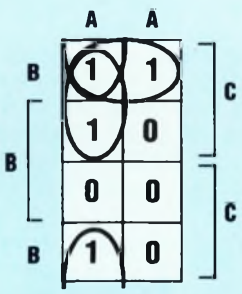
Die Schaltung dazu muß so aussehen:



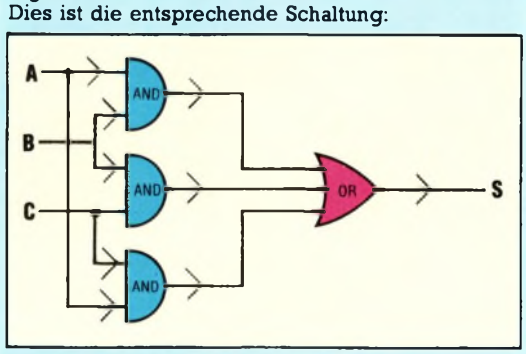
5) Wahrheitstabelle:

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Aus der k-Tafel:



ergibt sich der Ausdruck: S=A.B+A.C+B.C.
Dies ist die entsprechende Schaltung:



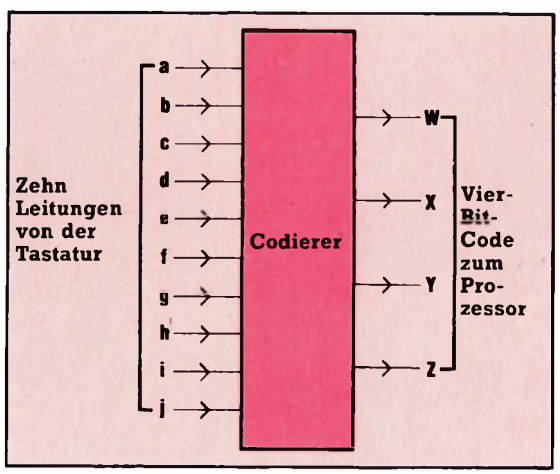
Leitungswechsel

Die CPU eines Computers übermittelt die Befehle an die Verarbeitungseinheiten in Form elektrischer Signale. Diese Signale werden vor dem „Abschicken" verschlüsselt und beim „Empfänger" wieder zurückübersetzt. Die dabei verwendeten Codier- und Decodierschaltungen sollen etwas gründlicher erläutert werden.

Der Prozessor schickt Anweisungen sowohl an interne (Akkumulator, ALU) als auch an externe Peripheriegeräte – etwa an den Drukker. Durch Verschlüsselung der Befehle ist dieser Datentransfer mit einem Minimum an Leitungen zwischen Prozessor und Peripherie möglich. Ein Beispiel dafür ist die Tastaturabfrage. Eine Tastatur hat eine bestimmte Anzahl von Ausgangsleitungen, von denen immer nur eine einzige eine Signalspannung führt. Das Tastatur-Signal kann daher auch als sechsstellige Binärzahl verschlüsselt werden – zur Übertragung sind dann nur noch sechs Leitungen erforderlich. Dazu kommen allerdings noch zwei Leitungen, eine für das Prüf-Bit und eine zweite für Fälle, in denen die Shift- oder Control-Taste mit einer anderen Taste zusammen gedrückt wird.

Vier Ausgänge

Um das Prinzip des dafür eingesetzten Codierers zu verdeutlichen, wählen wir das Beispiel einer einfacheren Tastatur mit nur zehn Tasten, auf der sich die Ziffern von 0 bis 9 eingeben lassen. Mit einem Drei-Bit-Wort hätten wir nur acht mögliche Bit-Kombinationen zur Verfügung, bei zehn Eingangsleitungen muß die Schaltung also vier Ausgänge haben. Die folgende Tabelle zeigt die entsprechenden Leitungen einer Zehner-Tastatur:

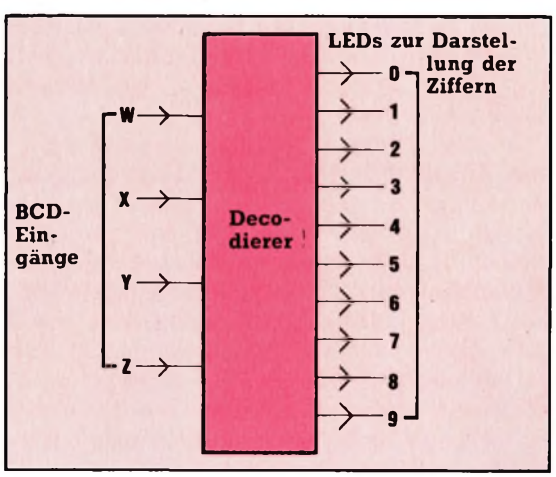


Weil jeweils nur eine der zehn Leitungen aktiv (High) sein kann, sieht die Wahrheitstabelle des Codierers folgendermaßen aus:

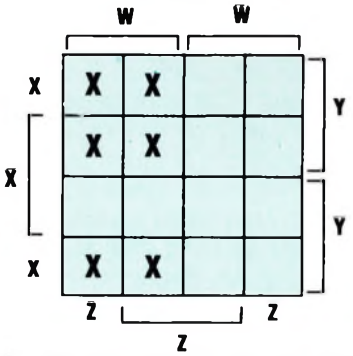
Dezimal	Eingänge										Ausgänge			
	a	b	c	d	e	f	g	h	i	j	W	X	Y	Z
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	1	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	0	0	0	0	1	0	1
6	0	0	0	0	0	0	1	0	0	0	0	1	1	0
7	0	0	0	0	0	0	0	1	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	1	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1	0	0	1

Das Rückübersetzen aus einem Code nennt man „Decodieren". Dabei bedienen nur wenige Leitungen (meist direkt über den Binärcode des Prozessors) die Ausgabe einer Vielzahl verschiedener Datenkanäle. Dieses Verfahren wird zum Beispiel häufig zur Drucker- oder Plotteransteuerung eingesetzt, ist aber auch für die Positionierung des Schreib/Lesekopfes in einer Diskettenstation oder für die Wahl eines bestimmten Ausgabekanals zu einem Peripheriegerät gebräuchlich.

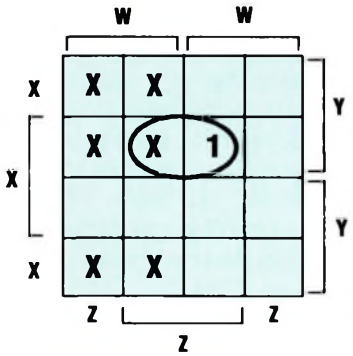
Die Arbeitsweise eines einfachen Decodierers aus AND-, OR- und NOT-Gattern läßt sich am besten mit einem Beispiel verdeutlichen: Der Decodierer soll einen BCD-Code (Binary-Coded-Decimal) so umwandeln, daß damit zehn einzelne LEDs entsprechend dem dezimalen Wert des Codes angesteuert werden können. Das wäre genau die Umkehrung der Aufgabenstellung des Codierers aus dem Tastatur-Beispiel.



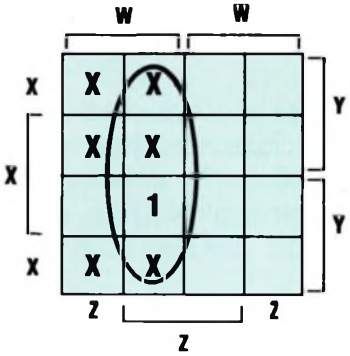
Im BCD-Code werden die dezimalen Ziffern 0 bis 9 durch je ein Vier-Bit-Wort dargestellt, der Decodierer hat also vier Eingänge. Damit sind 16 unterschiedliche Kombinationen von „Highs" in den Leitungen möglich. Da wir nur zehn Kombinationen brauchen, werden die restlichen sechs als „ungültige Eingabe" (X) betrachtet. Mit Booleschen Ausdrücken können wir die Abhängigkeit jedes der zehn Ausgänge vom Zustand der Eingabeparameter W,X,Y und Z einzeln angeben (Tabelle 2).



Wenn Sie die Ausgaben einzeln betrachten, finden sich weitere Möglichkeiten der Vereinfachung. Durch die Eintragung des Booleschen Ausdrucks für die Ausgabe der Zahl 3 ergibt sich eine verkürzte Darstellungsmöglichkeit: X.Y.Z.



Ähnlich wird im zweiten Beispiel mit dem Ausdruck für die Zahl 9 verfahren:

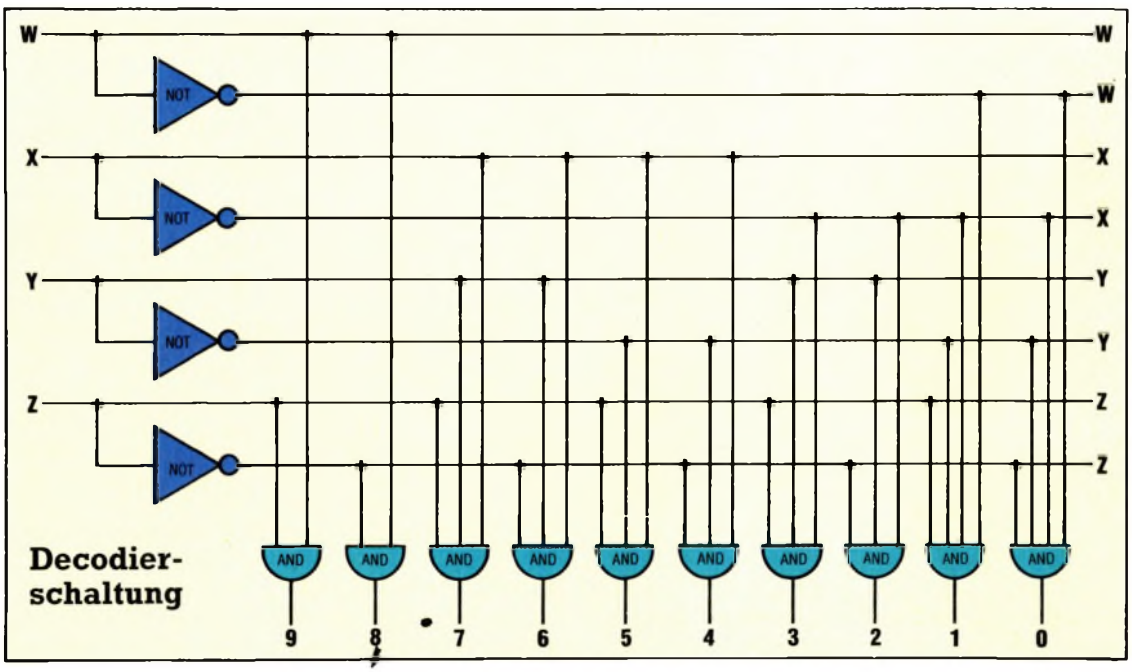


Die Schleife umschließt neben dem Ausdruck für die 9 drei Fälle von ungültiger Eingabe und steht für den Booleschen Ausdruck W.Z. Auf die gleiche Weise können noch einige andere Ausdrücke verkürzt werden.

Jetzt muß nur noch eine geeignete Schaltung zur Realisierung der zehn Booleschen Ausdrücke konstruiert werden. Da jede der vier Eingaben sowohl in normaler als auch in verneinter Form gebraucht wird, arbeitet man am besten mit acht parallelen Leitungen. Dann müssen für alle zehn Ausgänge nur Abzweigungen von den entsprechenden Leitungen hergestellt und mit AND-Gattern zusammengefaßt werden.

Übung 6

1) Entwerfen Sie eine Codierschaltung mit drei Eingängen, die bei Eingabe von 011,101,110 und 111 eine 1 und bei allen anderen Eingaben eine 0 ausgibt.
a) Zeichnen Sie dazu die Wahrheitstabelle.
b) Entwickeln Sie den Booleschen Ausdruck für die Ausgabe; vereinfachen Sie ihn.
c) Entwerfen Sie die Schaltung dazu.



Decodier-schaltung

Tabelle 1

Eingaben				BCD Zahl
W	X	Y	Z	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Tabelle 2

BCD Zahl	Boolescher Ausdruck
0	$\bar{W}.\bar{X}.\bar{Y}.\bar{Z}$
1	$\bar{W}.\bar{X}.\bar{Y}.Z$
2	$\bar{W}.\bar{X}.Y.\bar{Z}$
3	$\bar{W}.\bar{X}.Y.Z$
4	$\bar{W}.X.\bar{Y}.\bar{Z}$
5	$\bar{W}.X.\bar{Y}.Z$
6	$\bar{W}.X.Y.\bar{Z}$
7	$\bar{W}.X.Y.Z$
8	$W.\bar{X}.\bar{Y}.\bar{Z}$
9	$W.\bar{X}.\bar{Y}.Z$

Tabelle 3

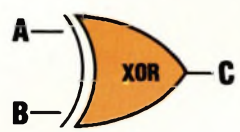
BCD Zahl	Boolescher Ausdruck
0	$\bar{W}.\bar{X}.\bar{Y}.\bar{Z}$
1	$\bar{W}.\bar{X}.\bar{Y}.Z$
2	$\bar{X}.Y.\bar{Z}$
3	$\bar{X}.Y.Z$
4	$X.\bar{Y}.\bar{Z}$
5	$X.\bar{Y}.Z$
6	$X.Y.\bar{Z}$
7	$X.Y.Z$
8	$W.\bar{Z}$
9	$W.Z$

Prüfroutinen

In diesem Abschnitt wollen wir den Weg zweier Baugruppen nachvollziehen – von der Planung eines Prüfbit-Generators und einer Vorrangschaltung führt er über die Wahrheitstabelle und deren Vereinfachung mit Boolescher Algebra bis zum kompletten Schaltplan.

Bevor wir uns mit der Erstellung der Baugruppen befassen, müssen wir ein weiteres Logikgatter genauer kennenlernen – das ausschließende OR-(XOR)-Gatter. Dieses Bauteil ist zwar schon früher erwähnt worden, seine Boolesche Algebra und das Schaltungssymbol wurden jedoch noch nicht erklärt:

Wahrheitstabelle			Schaltungssymbol
A	B	C	
0	0	0	
0	1	1	
1	0	1	
1	1	0	



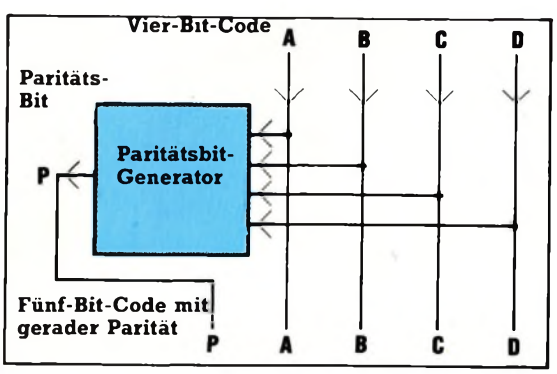
Boolesches Symbol =⊕

Die Wahrheitstabelle zeigt, daß sich der Ausgang C auf zwei Arten darstellen läßt:

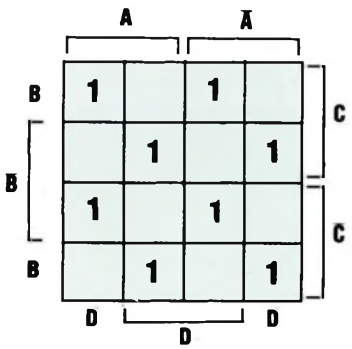
a) $C = A \oplus B = \overline{A}.B + A.\overline{B}$

b) $\overline{C} = \overline{A \oplus B} = \overline{A}.\overline{B} + A.B$

Der zweite Ausdruck ergibt sich aus den Fällen, in denen C nicht Eins ist. Das XOR-Gatter kommt im ersten Beispiel zum Einsatz:



Paritäts- oder Prüfbits sind ein wichtiger Faktor beim Entwurf von Datenübermittlungs-Systemen. Prüfbits werden einem Datencode hinzugefügt, so daß jede einzelne Codegruppe stets eine gerade Anzahl (bzw. nach Vereinbarung auch ungerade Zahl) von Einsen enthält. (Gerade Anzahl = „Even Parity", ungerade Anzahl = „Odd Parity".) Das Paritätsbit erlaubt die Prüfung der korrekten Datenübermittlung. Unsere Schaltung erzeugt das Prüfbit für eine Vier-Bit-Eingabe. Mit kleinen Änderungen kann sie auch als Prüfschaltung für eingehende Daten umgebaut werden. Die Wahrheitstabelle befindet sich am linken Rand dieser Seite. Hier die entsprechende k-Tafel:



Der symmetrische Aufbau der Tafel macht die Vereinfachung unmöglich, weil sich aus den Werten keine Gruppen bilden lassen. Der Ausdruck für P lautet:

$$P=\overline{A}.\overline{B}.\overline{C}.D+\overline{A}.\overline{B}.C.\overline{D}+\overline{A}.B.\overline{C}.\overline{D}+\overline{A}.B.C.D$$
$$+A.\overline{B}.\overline{C}.\overline{D}+A.\overline{B}.C.D+A.B.\overline{C}.D+A.B.C.\overline{D}$$

Die rot- und die blaugedruckten Ausdrücke können zusammengefaßt werden:

$$P=(\overline{A}.\overline{B}+A.B).(\overline{C}.D+C.\overline{D})$$
$$+(A.\overline{B}+\overline{A}.B).(\overline{C}.\overline{D}+C.D)$$

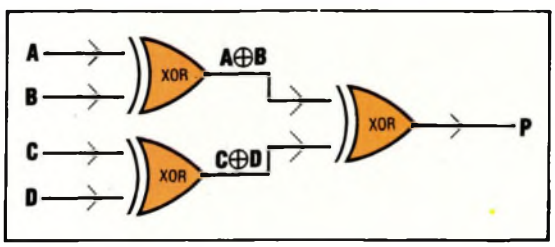
Hier läßt sich der Ausdruck für das XOR-Gatter (siehe oben) zur weiteren Vereinfachung einsetzen:

$P=(\overline{A \oplus B}).(C \oplus D) + (A \oplus B).(\overline{C \oplus D})$

Wenn der Inhalt jeder Klammer als Eingabe zum XOR-Gatter benutzt wird, reduziert sich der Ausdruck noch weiter:

$P=(A \oplus B) \oplus (C \oplus D)$

Das entspricht einer Kaskadenschaltung aus XOR-Gattern:



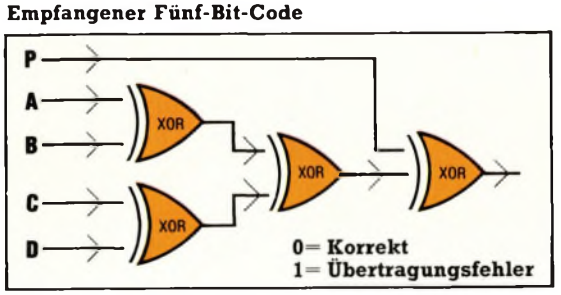
Durch ein weiteres XOR-Gatter wird die Schaltung zum Paritätsprüfer: Das Gatter vergleicht

Wahrheitstabelle

A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

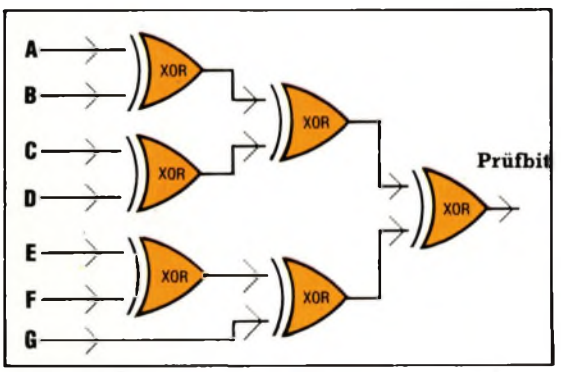
dazu das eingehende Prüfbit mit einem im Empfangsgerät erzeugten Paritätsbit.

Empfangener Fünf-Bit-Code



In der Praxis wird für die Datenübertragung meist der ASCII-Code verwendet, der aus sieben Datenbits und einem Paritätsbit besteht. Ein Prüfbit-Generator für ASCII hätte diese Schaltung:

Sieben-Bit-Daten



Die meisten Computer regeln ihren Datenfluß über ein System von Vorrangschaltungen. Die CPU kann durch das Signal eines Peripheriegerätes aufgerufen werden (Interrupt). Manchmal geschieht das von mehreren Peripheriegeräten gleichzeitig. Unsere Vorrangschaltung verbindet vier Peripheriegeräte und schaltet nur den Aufruf des Gerätes mit der höchsten Priorität an den Computer durch.

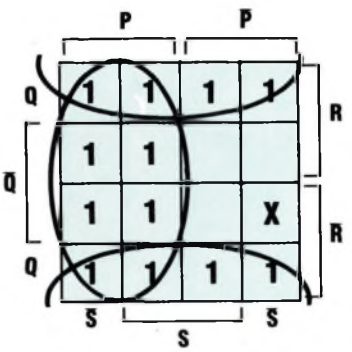
Für den Aufruf des Peripheriegerätes sind zwei Leitungen nötig. Eine dritte Ausgangsleitung gibt an, ob gerade ein Aufruf stattfindet. Die Geräte sollen P, Q, R und S heißen, wobei P die höchste und S die niedrigste Priorität hat. Die Ausgangsleitungen A und B melden das betreffende Peripheriegerät, Z ist die Rufleitung. Bedingungen, die unsere Vorrangschaltung ignorieren soll, sind in der Wahrheitstabelle jeweils mit X bezeichnet.

P	Q	R	S	A	B	Z
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Zum besseren Verständnis hilft die Betrachtung der letzten Zeile: Hier ruft P, das Gerät mit höchster Priorität, den Computer auf – alle Signale der niederwertigeren Geräte werden infolgedessen ignoriert.

Die drei Ausgangsleitungen müssen einzeln analysiert werden. Beginnen wir mit A, deren k-Tafel so aussieht:

Für A

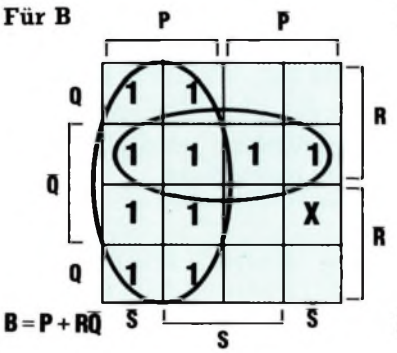


Der zu ignorierende Fall für die Ausgangsseite von A ist in der k-Tafel mit X bezeichnet. Anders sieht es auf der Eingangsseite aus: Wenn P = Eins und Q, R und S zu ignorierende Bedingungen sind, müssen alle acht Felder der k-Tafel ausgefüllt werden, in denen P = Eins ist. Mit der k-Tafel finden wir die Vereinfachung zu diesem Ausdruck:

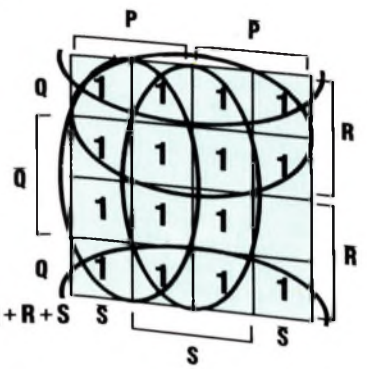
$$A = P + Q$$

Hier die k-Tafeln für Leitungen B und Z:

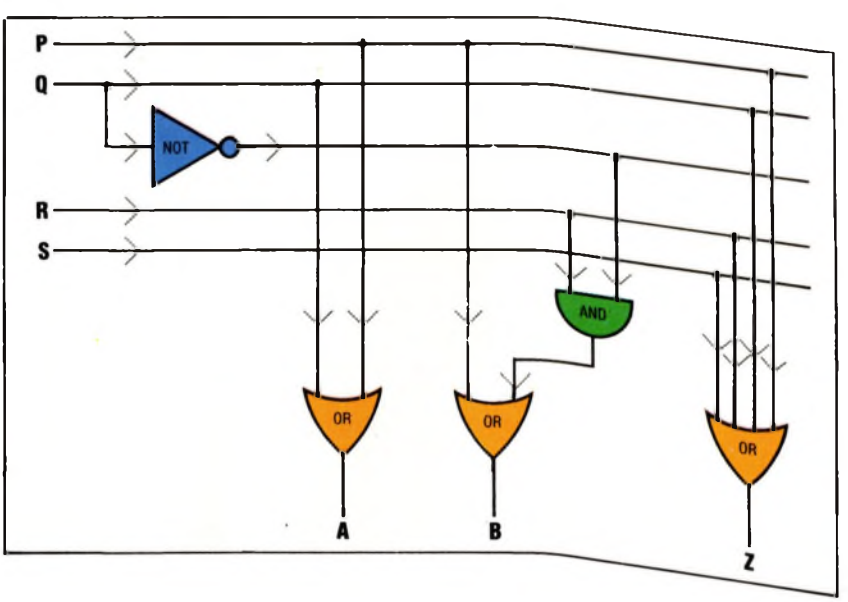
Für B



Für Z



Aus der Verbindung der drei Ausdrücke ergibt sich dieser Schaltungsaufbau:



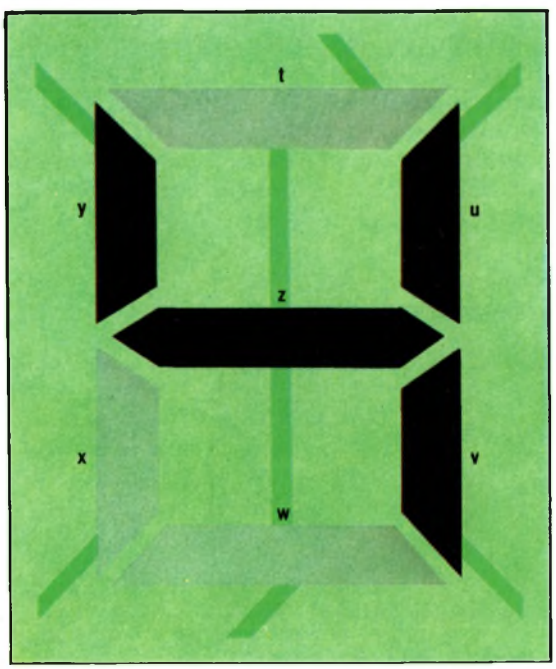
Zahlenspiele

In diesem Teil des Logik-Kurses beschäftigen wir uns mit der Konstruktion eines „Binär-Sieben-Segment-Wandlers". Diese Schaltung setzt die binären Signale des Computers in lesbare Ziffern um.

Unsere Schaltung eignet sich für die Umsetzung aus dem BCD-Code. BCD ist ein Akronym für „Binary Coded Decimal", also „Binär Codierte Dezimalziffern". Dieser Code ordnet jeder Ziffer von 0 bis 9 einen bestimmten Vier-Bit-Wert zu:

BCD	Dezimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010 to 1111	bis ungültig

Eine Sieben-Segmentanzeige zeigt Dezimalziffern durch das Einschalten bestimmter Leuchtdioden, bei Flüssigkristallanzeigen (LCDs) wird die Spannung an einzelnen Streifen umgepolt. In der Abbildung wurden die Einzelsegmente mit den Buchstaben t bis z bezeichnet. Die Tabelle weiter unten zeigt, welche Kombination von Segmenten zur Darstellung einer Dezimalziffer (0 bis 9) eingeschaltet sein muß.



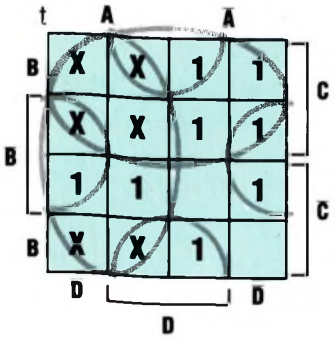
Die LCD-Anzeige besteht aus sieben separat schaltbaren Streifen, die hier mit den Buchstaben t bis z bezeichnet wurden.

Die vorhandenen Daten genügen bereits zur Aufstellung der Wahrheitstabelle des Wandlers. Das Bild auf der nächsten Seite zeigt, welche Eingabe die Aktivierung einer bestimmten Segmentkombination auslöst. Der binäre Eingabewert 0100 (dezimal 4) schaltet beispielsweise die Segmente u,v,y und z ein, die Eingabe von 1000 (dezimal 8) aktiviert alle Segmente gleichzeitig. Die Wahrheitstabelle:

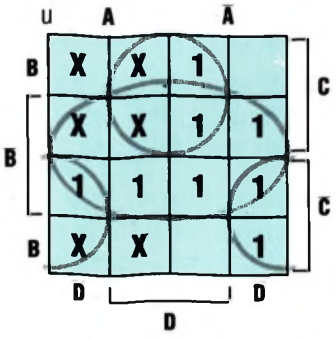
Dezimal	Eingänge				Ausgänge						
	A	B	C	D	t	u	v	w	x	y	z
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
	1	0	1	0	X	X	X	X	X	X	X
	1	0	1	1	X	X	X	X	X	X	X
	1	1	0	0	X	X	X	X	X	X	X
	1	1	0	1	X	X	X	X	X	X	X
	1	1	1	0	X	X	X	X	X	X	X
	1	1	1	1	X	X	X	X	X	X	X

Eine Analyse der Wahrheitstabelle ist leider unmöglich – die Ausgänge t,u,v,w,x,y und z müssen also jeder für sich vereinfacht werden. Dazu brauchen wir sieben verschiedene Karnaugh-Tafeln, die jeweils die Bedingung „Ignorieren" (X) enthalten. Sie sind auf der nächsten Seite abgedruckt. Durch Zusammenfassung der eingekreisten Gruppen lassen sich diese Ausdrücke vereinfachen, und Faktorisierung führt in einigen Fällen zu einer weiteren Reduktion der Ausdrücke.

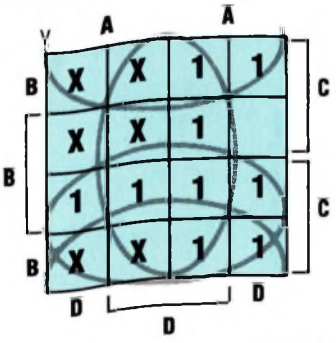
Die erste k-Tafel enthält alle Ausdrücke, für die die „t"-Balken benötigt werden – es gibt nur zwei Ziffern, in denen „t" nicht benötigt wird. Sind die Formeln für jeden Ausgang vereinfacht, geht es mit dem eigentlichen Schaltungsentwurf weiter.



t=C+A+B.D+B.D

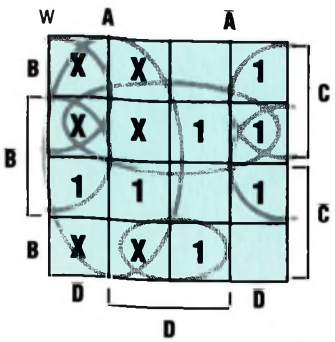


u=B+C.D+C.D

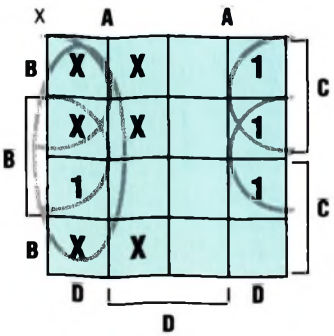


v=D+C+B

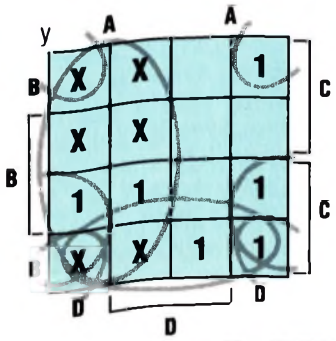
Aus den Tafeln können Sie ersehen, welche der Segmente für die einzelnen Ziffern eingeschaltet sein müssen. Für die „1" werden nur die Segmente „u" und „v" gebraucht.



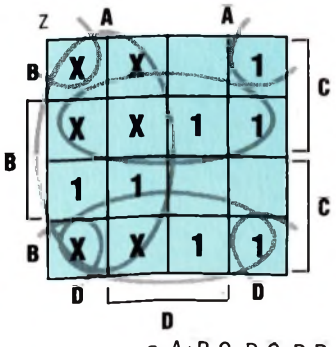
w=A+C.D+B.C+B.D+B.C.D.



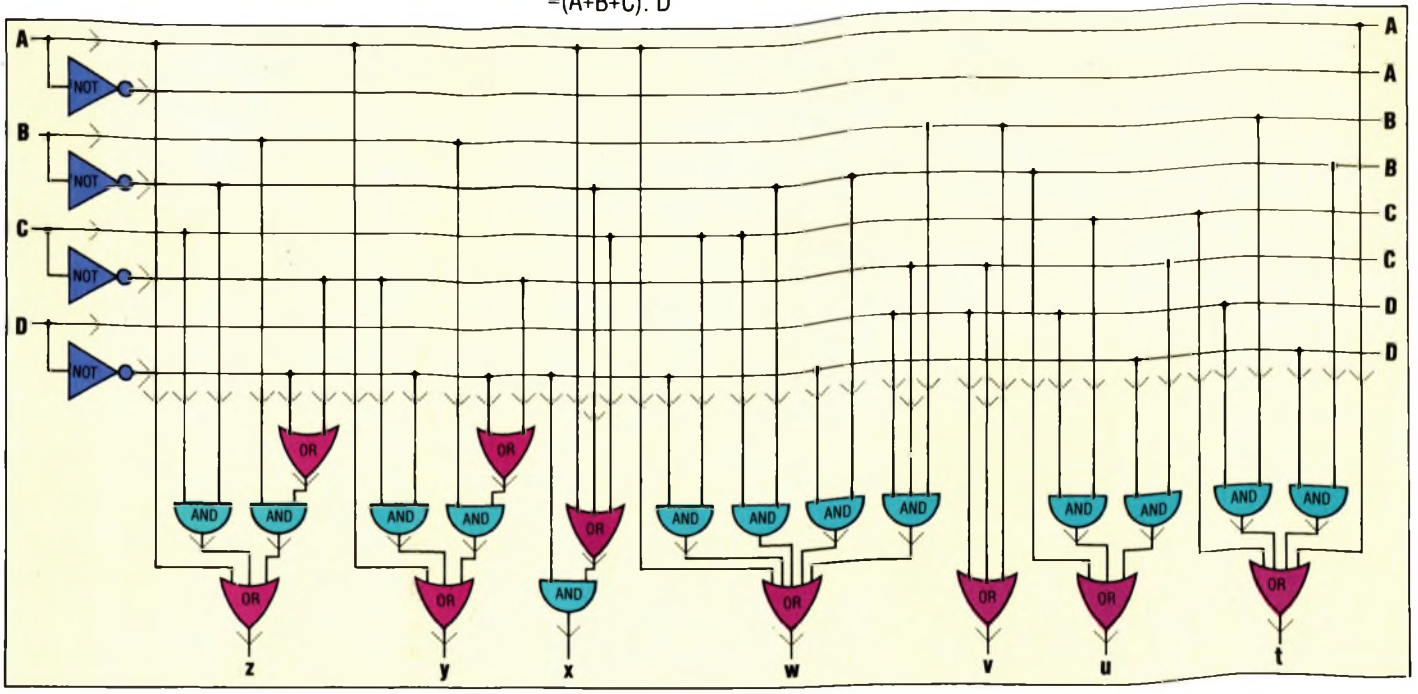
x=C.D+A.D+B.D
=(A+B̄+C). D



y=A+B.C+B.D+C.D
=A+B.(C+D)+C.D



z=A+B.C+B.C+B.D
=A+B.C+B.(C+D)

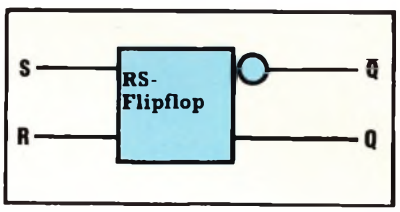


Die hier angegebene Schaltung kann nur eine Sieben-Segment-Anzeige steuern, Anzeigefelder bestehen aber meist aus acht bis zehn Einzelziffern. Trotzdem brauchen Sie dank des Multiplex-Verfahrens auch für mehrstellige Anzeigen nur einen Wandler, der die einzelnen Sieben-Segment-Anzeigen nacheinander steuert. Läuft dieser Vorgang schnell genug ab, erscheint die Anzeige völlig stabil – durch die hohe Schaltgeschwindigkeit läßt sich nicht erkennen, daß jede Anzeige immer nur für kurze Zeit aktiv ist.

Vom Flip zum Flop

Entgegen den zuvor erklärten Schaltkreisen produzieren sequentielle Schaltungen auf einen einzigen Impuls hin eine stetige Kette von Ausgangssignalen. Als Beispiel wollen wir die Funktionsweise eines RS-Flipflops untersuchen.

Es gibt verschiedene Flipflop-Schaltungen, die jedoch alle ähnlich arbeiten. Ein RS-Flipflop hat zwei Ein- und zwei Ausgänge.

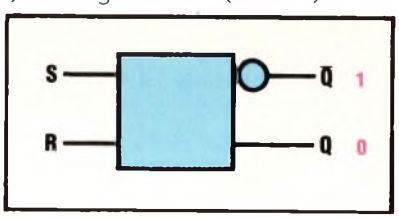


Die beiden Ausgänge Q und $\bar{Q}$ weisen immer einen gegensätzlichen Zustand auf, das bedeutet

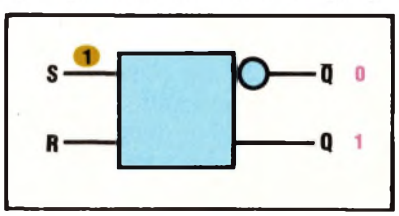
wenn Q = 1 dann $\bar{Q}$ = 0 (Set-Zustand)
wenn Q = 0 dann $\bar{Q}$ = 1 (Reset-Zustand)

Wenn sich der Flipflop zu Beginn im RESET-Zustand befindet, schaltet ihn ein Impuls am S-Eingang auf den SET-Zustand um.

1) Anfangszustand (RESET)

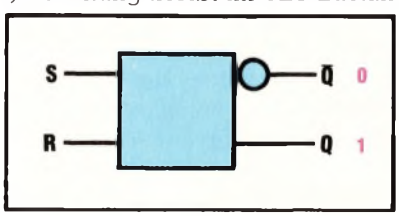


2) Impuls auf dem SET-Eingang



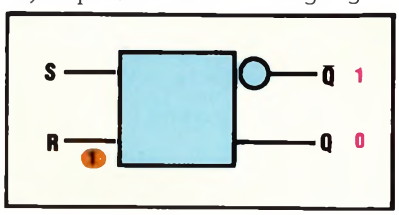
Die Schaltung bleibt auch nach dem Ende des Impulses auf Eingang S im Set-Zustand.

3) Schaltung bleibt im SET-Zustand

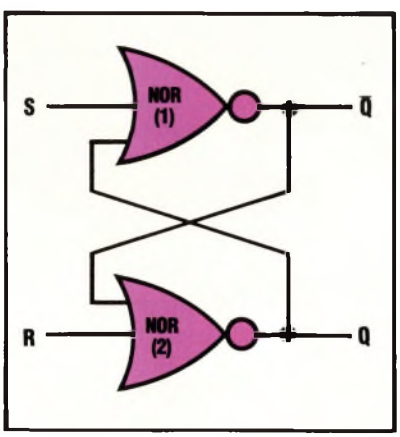


Ein weiterer Impuls auf Eingang R versetzt die Schaltung zurück in den alten RESET-Zustand.

4) Impuls auf dem R-Eingang

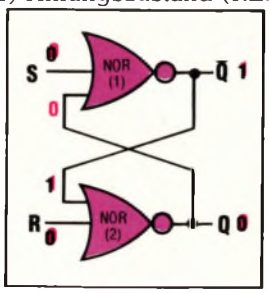


Wie sieht nun die Schaltungslogik aus, die das Flipflop-Verhalten ermöglicht? Es gibt mehrere Verfahren, eine Flipflop-Schaltung aufzubauen: die Verbindung zweier NAND-Gatter etwa oder, wie in dieser Zeichnung, das Zusammenschalten zweier NOR-Gatter, bei denen die beiden Ausgänge mit jeweils einem Eingang des anderen Gatters verbunden sind. Die „Speicherfähigkeit" der Schaltung beruht auf dieser Rückkopplung.

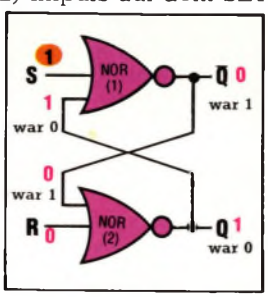


Versuchen wir, das SET- und RESET-Verhalten des Flipflops aus der Schaltung zu erklären: Angenommen, der Flipflop ist im RESET-Zustand. Ohne Eingangsimpulse bleibt er dann in diesem Zustand stabil (ein NOR-Gatter gibt nur dann eine 1 aus, wenn beide Eingänge auf 0 liegen). Ein Impuls auf Eingang S kippt den stabilen Zustand und schaltet den „NOT Q"-Ausgang auf 0. Der Ausgang ist mit dem Eingang des zweiten NOR-Gatters verkoppelt, und der Ausgang Q dieses Gatters schaltet daher auf 1. Ist der Impuls am ersten NOR-Gatter (1) noch vorhanden, liegen beide Eingänge auf 1, und der Ausgang des Gatters (1) bleibt weiter auf 0. Damit hat sich die Schaltung im SET-Zustand stabilisiert.

1) Anfangszustand (RESET)

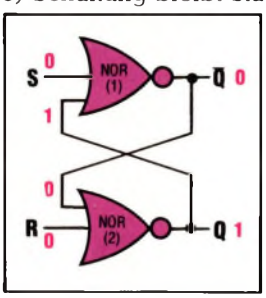


2) Impuls auf dem SET-Eingang

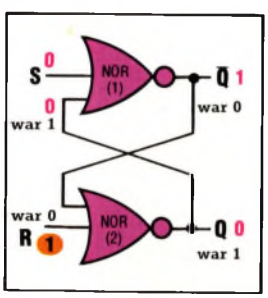


Die Schaltung bleibt auch dann im neuen Zustand, wenn der Eingangsimpuls vorbei ist. Erst ein Signal auf dem R-Eingang destabilisiert sie kurz, führt dann aber sofort wieder zu einem dauerhaften RESET-Status.

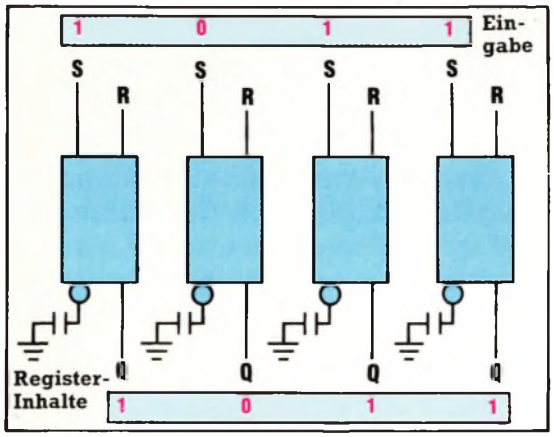
3) Schaltung bleibt stabil (SET)



4) Impuls auf dem RESET-Eingang

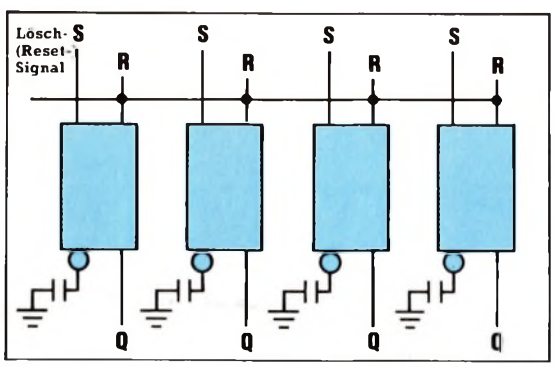


Der Microprozessor Ihres Computers besteht größtenteils aus einer Vielzahl von Speichern bzw. Registern wie dem Akkumulator, den Befehls- und Index-Registern. Die meisten Register können Acht-Bit-Wörter speichern – also achtstellige Binärzahlen mit dezimalen Werten von 0 bis 255. Diese Speicherung wird durch eine Kette aus acht Flipflops möglich. Um etwa die Binärzahl 1011 darin zu speichern, muß das entsprechende Bitmuster auf die S-Eingänge gegeben werden.



Bei dieser Anordnung ist der „NOT Q"-Ausgang unbenutzt. Da das Bitmuster über den S-Eingang zugeführt wird, steht es am Q-Ausgang zur Verfügung. Zum Überschreiben einer im Register gespeicherten Zahl mit einem neuen Wert – etwa 0110 – reicht es nicht aus, einfach nur das neue Muster über den S-Eingang einzugeben. Dabei würden in diesem Fall die beiden äußeren Einsen erhalten bleiben, und der falsche neue Wert wäre 1111.

Die Lösung des Problems liegt darin, alle Flipflops vor einer neuen Belegung zu löschen. Da dies bei allen Registerstellen gleichzeitig passieren muß, werden die Flipflops verbunden, so daß zum Löschen nur ein einziges Signal nötig ist.



Im nächsten Kursabschnitt werden Sie weitere sequentielle Schaltungen kennenlernen, unter anderen auch den D- und den JK-Flipflop.

Übung 7

1) Warum bezeichnet man Flipflops als „bistabil"?

2) Beim Einschalten eines Computers befindet sich ein Flipflop im folgenden Zustand:
$Q = 0, \bar{Q} = 0, S = 0, R = 0$

a) Ist dies ein stabiler Zustand?

b) Falls nicht, in welchen Zustand wird der Flipflop übergehen?

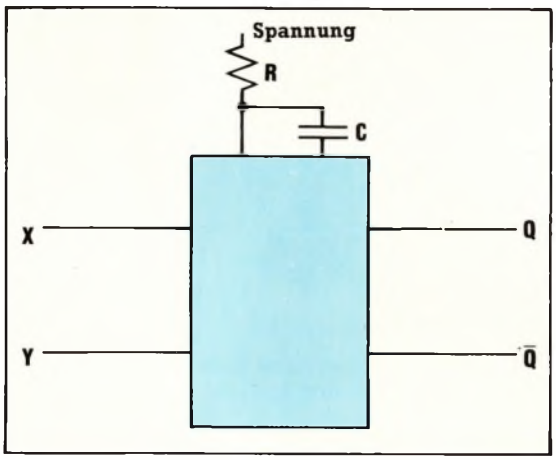
c) Kann der Flipflop auch in einen anderen Zustand als den in Ihrer Lösung zu b) übergehen? (Tip: Beginn beim anderen Gatter!)

d) Was muß unternommen werden, damit sich nach dem Einschalten eines Computers alle Register in einem vorhersehbaren Zustand befinden?

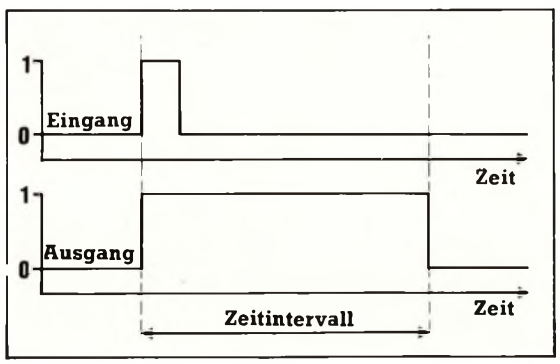
Taktvoll

Zur exakten Steuerung eines Computers ist genaues „Timing" unabdingbar. In diesem Kursabschnitt wollen wir Ihnen monostabile Kippschaltungen und zwei spezielle Flipflops vorstellen.

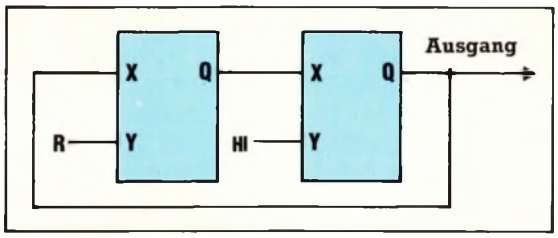
Mit monostabilen Schaltungen lassen sich logische Abläufe in exakten Zeiteinheiten steuern. Empfängt die Kippschaltung einen Impuls, wird der Ausgang für eine bestimmte Zeit auf 1 (HI) gesetzt, bevor er in den Ruhezustand (LO) zurückkehrt. Hier das Beispiel einer Kippschaltung:



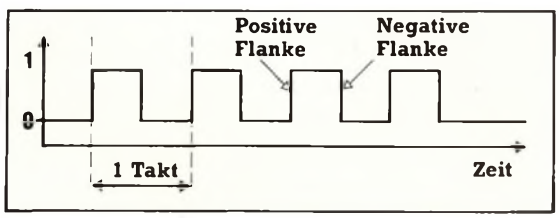
Der beschriebene Ablauf wird entweder durch Umschalten des X-Einganges von HI auf LO oder durch Umschalten des Y-Einganges von LO auf HI gestartet. Die Werte von Widerstand (R) und Kondensator (C) entscheiden über das Schaltintervall. Diese Kurven stellen den zeitlichen Zusammenhang der Ein- und Ausgangszustände dar:



Das HI-Intervall am Ausgang läßt sich etwa zur Steuerung eines Cassettenrecorder-Schrittmotors oder zur Verzögerung der Ubertragung eines Bits einsetzen. Mit zwei monostabilen Kippschaltungen kann ein Oszillator aufgebaut werden, der in gleichbleibenden Zeitabständen zwischen HI- und LO-Intervallen hin- und herschaltet: Am Ausgang ergibt sich eine

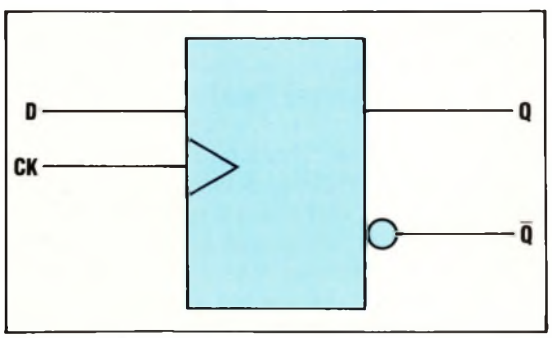


Rechteckspannung. Die Zeit zwischen zwei HI-Zuständen wird als „Zyklus" bezeichnet. Gewöhnlich liegen diese Zykluszeiten im Bereich von einer Millionstelsekunde. Der Maschinentakt ist der „Herzschlag" Ihres Computers, der alle Abläufe in der CPU dirigiert. Die zwei Flanken der einzelnen Rechteck-Impulse haben verschiedene Namen, die wir in der Zeichnung angeben:



Als nächstes sollen Sie zwei weitere Flipflop-Typen kennenlernen, die jedes Umschalten mit den Taktzyklen eines Oszillators koordinieren können.

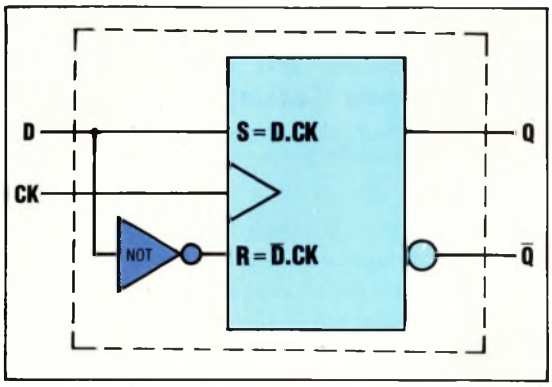
D-Flipflops haben einen logischen (D) und einen Takteingang (CK = clock = Uhr):

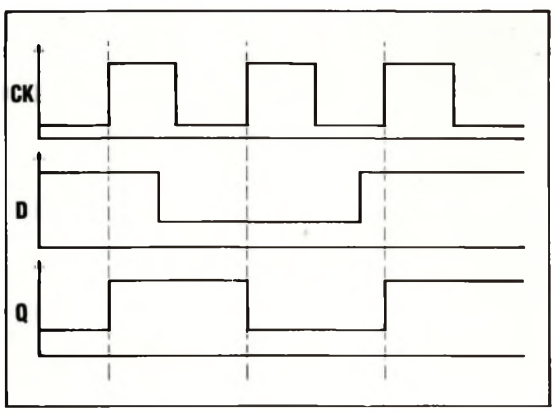


Der D-Flipflop baut auf dem bekannten RS-Typ aus unserem letzten Kursabschnitt auf. Durch den zusätzlichen Clock-Eingang kann jedoch der Eingang der Schaltung zeitweilig verriegelt werden.

Der Ausgangszustand von Q wird am Beginn jedes neuen Taktzyklus festgelegt. Liegt zu

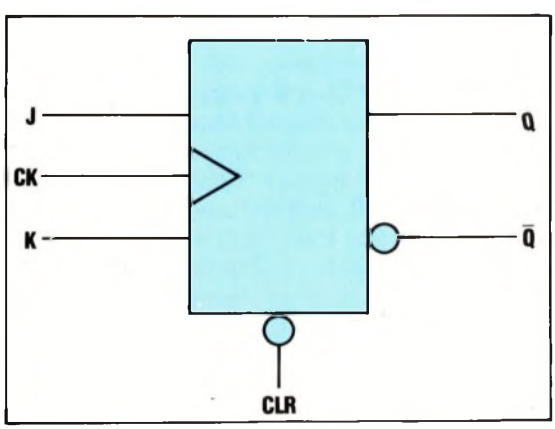
diesem Zeitpunkt ein HI auf Eingang D, so wird auch Q auf HI gesetzt, ist D aber LO, geht auch Q auf LO.



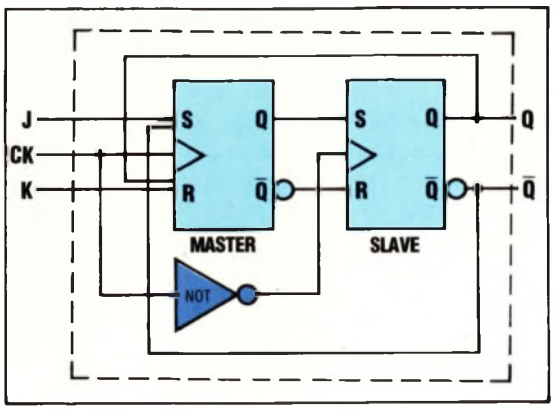


Aus der Zeichnung wird deutlich, daß Ausgang Q nur veränderlich ist, wenn der Eingang CK gerade von LO nach HI springt.

Der JK-Flipflop wirkt nach dem „Master-Slave"-Prinzip: In diesem Schaltungstyp arbeitet einer der zwei zusammengehörigen RS-Flipflops abhängig vom anderen. Bei der Veränderung des Eingangswertes wird das HI oder LO in einem Flipflop zwischengespeichert. Am Ausgang steht bis zum nächsten Taktzyklus der Eingangswert des vorherigen Taktes. Nützlich ist dies etwa bei der in vielen Prozessoren häufigen Verschiebung von Bits um eine Stelle nach links oder rechts. Hier das Schaltbild:

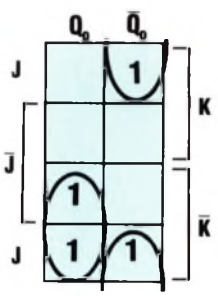


Das nächste Bild zeigt die Schaltung von zwei RS-Flipflops. Einer ist „Master" (Herr), der andere „Slave" (Sklave). Wenn CK HI ist, geht das Eingangssignal zum Master-Flipflop. Bei einem LO auf CK wird das Eingangssignal wegen der „Triggerung" mit der positiven Flanke auf den Slave-Flipflop geleitet. Es ist jeweils nur ein Flipflop aktiv, während der andere als Zwischenspeicher dient:



Rechts ist eine Tabelle mit den Schaltzuständen des JK-Flipflops abgedruckt. Q_0 bezeichnet dabei den Ausgangswert während des vergangenen Taktzyklus. Wenn sowohl am J- als auch am K-Eingang ein HI liegt, ändert der Ausgang bei jedem Taktzyklus seinen Zustand. Dieses als „toggling" bezeichnete Verhalten entsteht durch Rückkopplung vom Slave- zum Master-Flipflop. Stellt man die k-Tafel auf, müssen Q_0, J und K als Eingangsvariablen aufgefaßt werden.

Q_0	J	K	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



Aus der Tafel ergibt sich der Ausdruck

$$Q = \overline{Q}_0 \cdot J + Q_0 \cdot \overline{K}$$

Diese Gleichung wird als die „Charakteristik" des JK-Flipflops bezeichnet.

Lösungen der Übung 7

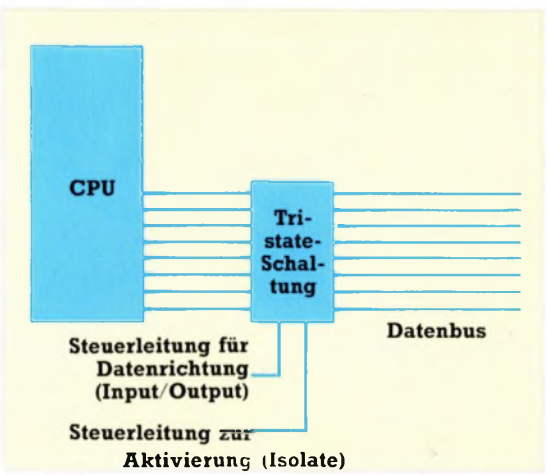
1) Einen Flipflop nennt man bistabil, weil er in zwei Zuständen stabil bleibt – wenn Q=1 und $\overline{Q}$=0 oder Q=0 und $\overline{Q}$=1 ist.
2) a) Ist kein stabiler Zustand.
b) Wird der obere Eingang zuerst berücksichtigt, geht der Flipflop auf RESET. Auf SET schaltet er, wenn zuerst der untere Eingang berücksichtigt wird.
c) Ja (siehe auch Antwort b).
d) Um den Zustand aller Register nach dem Einschalten festzulegen, müssen sie bei dem Systemstart entsprechend auf SET oder RESET gesetzt werden.

Bits im Bus

Diese Folge des Logik-Kurses befaßt sich mit dem System der Datenübertragung zwischen Prozessor und Speicher. Sie erfahren, was es mit Adressen, Datenbussen, Adreßregistern und Tri-State-Schaltungen auf sich hat.

Jeder Speicherplatz in einem Heimcomputer besteht normalerweise aus acht Bits. Diese acht Bits können gleichzeitig über acht Leitungen zur CPU gelangen, wo sie dann entsprechend den Anweisungen des Programms weiterverarbeitet werden. Aber auch der „Gegenverkehr" – von der CPU zum Speicher – ist möglich. Der Maschinencodebefehl LDA $1234 bewirkt etwa, daß der Wert von Speicherplatz $1234 an die CPU geschickt wird. STA $1234 hingegen sendet einen Wert von der CPU über den Datenbus zum Speicherplatz $1234.

Damit beides funktioniert, muß der Datenbus die Übertragung in beiden Richtungen erlauben. Manchmal ist es aber auch nötig, die CPU vom Datenbus zu trennen. Jede Busleitung kann deshalb in drei Zustände versetzt werden: INPUT, OUTPUT oder ISOLATE. Zum Wechseln zwischen diesen Zuständen verfügt jede Busleitung über eine sogenannte „Tri-State"-Schaltung.



Acht Tri-State-Schaltungen sind in einem IC vereinigt. Auf der kleinen Zeichnung sehen Sie, wie die Schaltung den Datenbus mit der CPU verbindet. Außerdem sind die Steuerleitungen für die Datenrichtung und die Aktivierung eingezeichnet, mit denen der gewünschte Betriebszustand angewählt wird. Solche Schaltungen werden auch beim Anschluß von Peripheriegeräten zur Ein- und Ausgabe auf den Datenbus verwendet.

Bestimmte Speicherplätze müssen immer über ihre Adresse abgefragt werden. Dazu trägt jeder Speicherplatz im ROM oder RAM als Namen eine eindeutige Ziffernkombination. Wie geht nun der Datentransfer vor sich?

Die meisten Heimcomputer haben neben dem Datenbus eine zweite Verbindung zwischen CPU und Speicher, den „Adreßbus". Oft hat der Adreßbus nicht nur acht, sondern 16 Leitungen, mit denen bis zu 65 536 einzelne Adressen – also 64 KByte – angesteuert werden können (2^{16}=65 536). Den gesamten Speicher muß man sich aufgeteilt in Sektionen mit jeweils 256 Speicherplätzen denken. Die acht niederwertigen Bits bestimmen den Speicherplatz innerhalb einer der Sektionen, die mit den acht höherwertigen Bits gewählt werden.

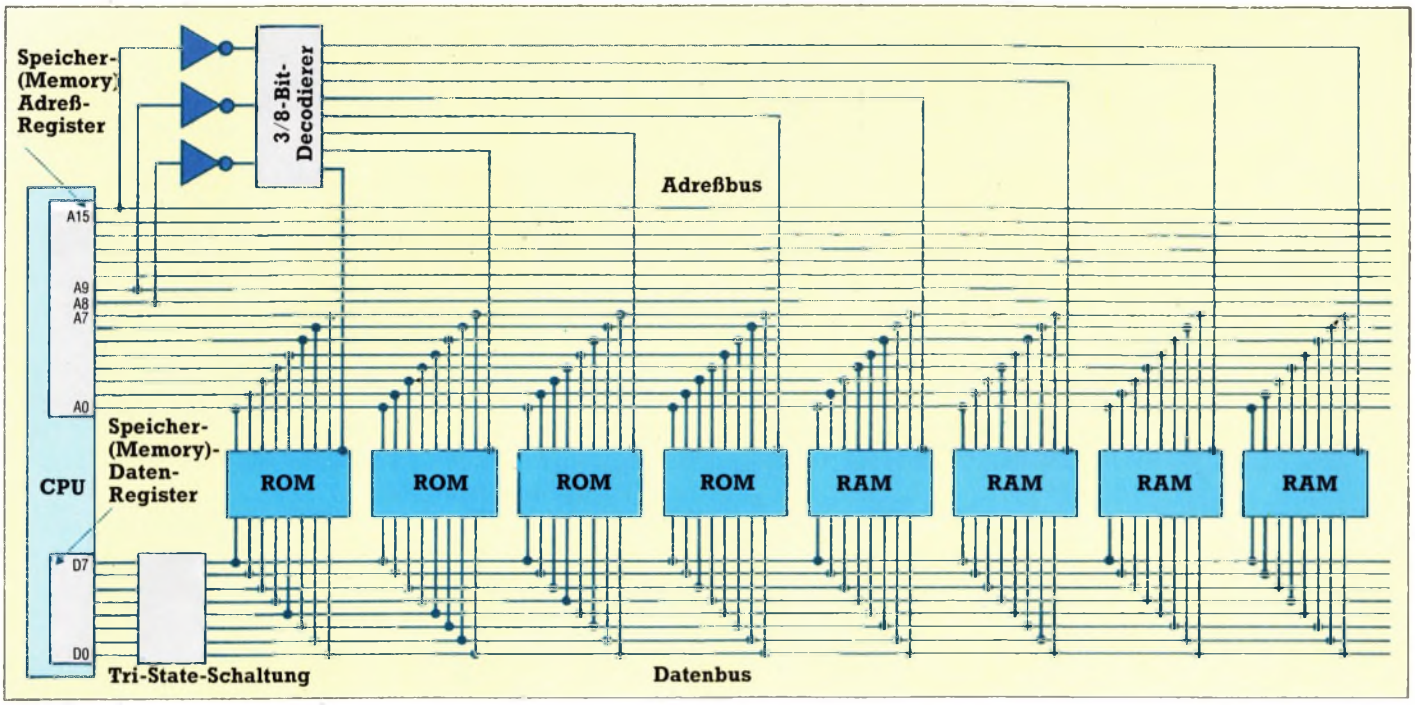
Am vereinfachten Beispiel eines Computers mit zwei KByte Speicher können wir sehen, wie das Ansprechen einer bestimmten Adresse vor sich geht. 256 Speicherplätze pro Sektion bedeutet, daß der Computer über insgesamt acht Sektionen verfügt. Für unser Beispiel gehen wir davon aus, daß der Speicherbereich zur Hälfte aus RAM-Bytes und zur Hälfte aus ROM besteht.

Memory Address Register

Die Adresse des benötigten Speicherplatzes findet sich in einem speziellen 16-Bit-Register der CPU, dem „Memory Address Register" oder „MAR". Da die acht niederwertigen Bits nur den Speicherplatz innerhalb einer Sektion definieren, können sie gleichzeitig mit allen Sektionen verbunden bleiben. Zur Auswahl eines bestimmten Moduls brauchen wir dann nur noch drei weitere Bits (2^3=8). Dieser Drei-Bit-Code muß auf acht Ausgangsleitungen – für jede Sektion eine – verteilt werden.

Die Zeichnung rechts oben zeigt, wie die Speicher-Sektionen über den Daten- und Adreßbus mit der CPU verbunden sind. Jede Sektion hat eine Zuleitung vom Drei(Bit)-auf-Acht(Leitungen)-Decodierer. Mit den drei höherwertigen Adreßbits wird die entsprechende Sektion angewählt.

Sie wissen jetzt, wie ein bestimmter Speicherplatz ausgewählt und die Daten daraus übertragen werden. Aber wie führt die CPU nun einen Befehl aus? Üblicherweise sind Maschinenprogramme in aufeinanderfolgenden Speicherplätzen abgelegt. Die Befehle können dabei auch zwei oder drei Speicherplätze belegen. Die Anweisung „ADD $13FF" bedeutet

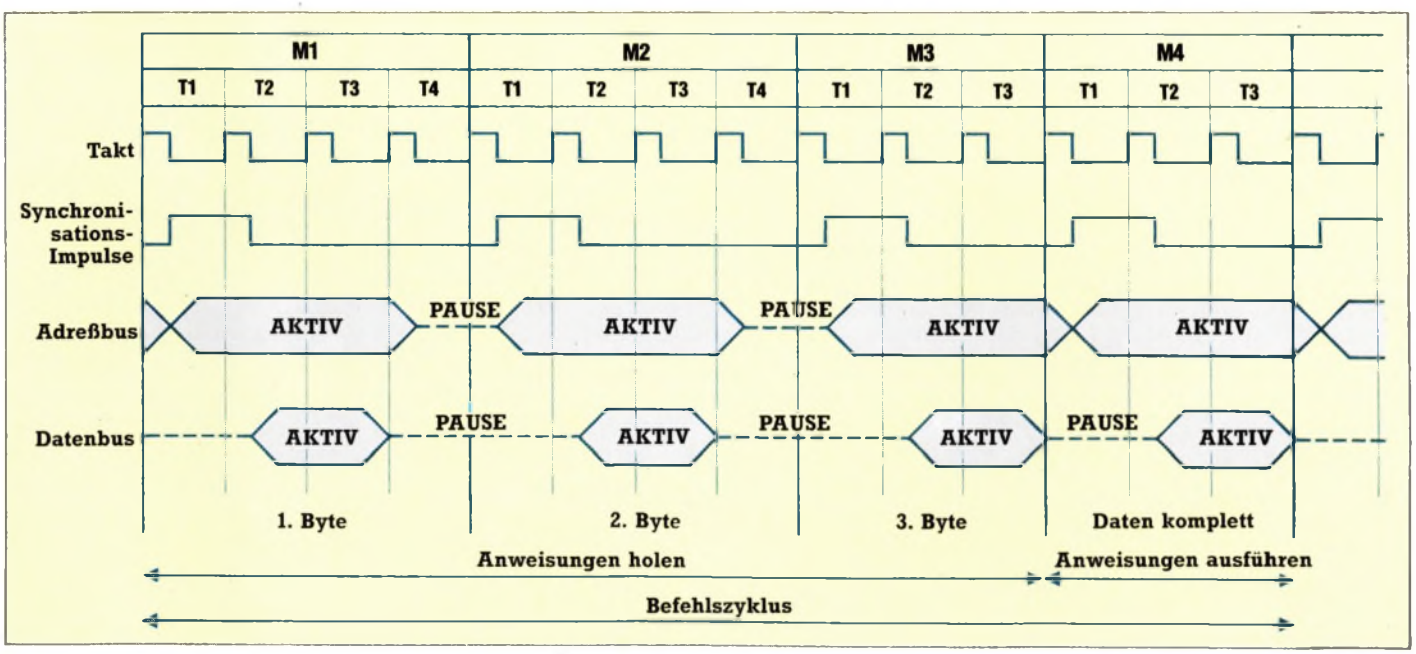


etwa „addiere den Inhalt des Speicherplatzes mit der hexadezimalen Adresse $13FF zum Akkumulator". Dafür wären drei Byte nötig: eines für den Befehl „ADD" und zwei für die 16-Bit-Adresse $13FF.

Bevor die Anweisung ausgeführt werden kann, muß sie erst einmal aus dem Speicher geholt werden. Drei Adressen müssen abgefragt werden, um die Bytes über den Datenbus zur CPU zu schaffen. Danach steht der komplette Befehl in einem Register der CPU, muß aber noch entschlüsselt und natürlich ausgeführt werden.

Computerhersteller veröffentlichen die Charakteristiken der verwendeten Prozessoren in Form von Zeitdiagrammen, welche die Abfolge der unterschiedlichen Aktionen des Computers darstellen. Die Überwachung des Ablaufs besorgt ein Taktgenerator. In unserem Beispiel wird der Adreßbus mit der positiven Flanke, der Datenbus mit der negativen Flanke eines Synchronisations-Impulses aktiviert. Der Synchronisations-Impuls selbst wird von der positiven Flanke des jeweils ersten Zeittaktes in einem Maschinenzyklus erzeugt. Die einzelnen Zyklen haben unterschiedliche Dauer, weil der Prozessor länger zur Entschlüsselung eines Maschinenbefehls als zur Verarbeitung eines numerischen Byte braucht. Die Entschlüsselung geht jedoch vor, weil erst der Befehl die genaue Anzahl der zu verarbeitenden Bytes angibt.

Die Abarbeitung eines Maschinenbefehls geht in zwei Phasen vor sich: Während der „Hol-Phase" wird über den Adreßbus zuerst der Befehl selbst von seinen Speicherplätzen zur CPU transportiert. Wenn sich der Befehl auf die Verarbeitung anderer Speicherinhalte bezieht, bleiben Daten- und Adreßbus in der darauf folgenden „Ausführungsphase" weiter in Aktion, denn auch die zu verarbeitenden Datenbits müssen nacheinander zum Prozessor geschafft werden.



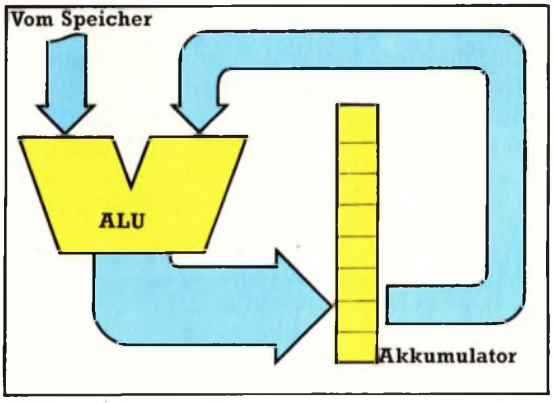
Logisches Finale

In jedem Computer wird der Datenfluß von einer zentralen Steuereinheit, der CPU, dirigiert. Die CPU ist auch für die Ausführung der Programmbefehle sowie für arithmetische und logische Berechnungen zuständig.

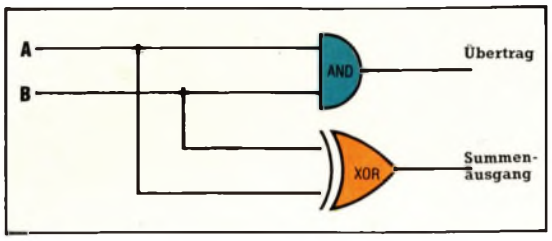
Das Arbeitsfeld der ALU (Arithmetik- und Logikteil der CPU) gliedert sich in zwei Gebiete: Rechenfunktionen, also Addition, Subtraktion und Inkrementierung (das schrittweise Erhöhen einer Zahl um den Wert Eins), und die Ausführung der drei wichtigsten Logikoperationen OR, XOR und AND.

Einige dieser Funktionen, etwa die Addition, beziehen sich auf zwei Operanden (Zahlen, die verarbeitet werden sollen), andere, wie die Inkrementierung, haben nur einen Operanden. Im letzteren Fall wird der benötigte Operand im Akkumulator, einem speziellen Register der CPU, zwischengespeichert. Bei zwei Operanden wird der fehlende aus dem Hauptspeicher abgerufen. Die beiden Zahlen werden nun durch den Schaltkreis der ALU „befördert", und die gewünschte Operation kann ausgeführt werden. Nach der Verarbeitung der Werte wird das Ergebnis wiederum in den Akkumulator geschrieben.

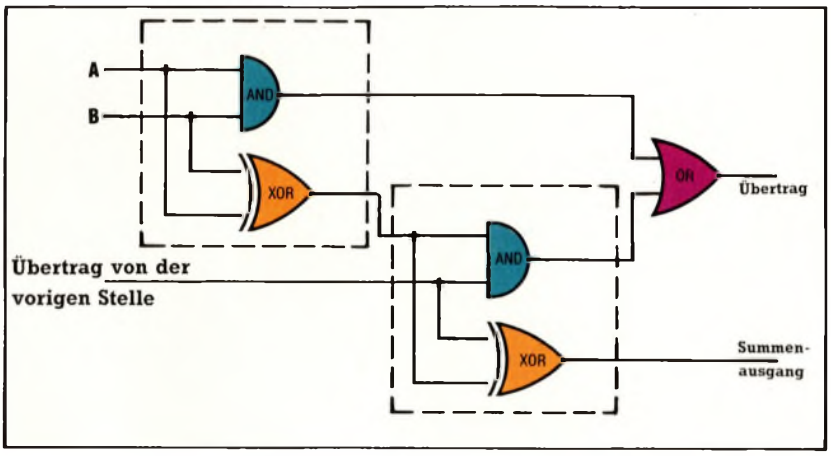
Diese Zeichnung verdeutlicht den Weg der Daten beim Passieren der ALU:



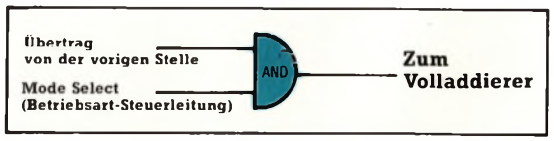
Zahlenwerte im Akkumulator und im Speicher werden mit acht Bits dargestellt. Diese acht Bits werden „parallel" verarbeitet, also alle gleichzeitig. Um die Arbeitsweise der ALU darzustellen, wollen wir eine Ein-Bit-Schaltung entwerfen, die alle sechs ALU-Funktionen ausführt. Das Bit des ersten Operanden nennen wir A, das des zweiten B. Grundlage unserer Ein-Bit-ALU ist ein Volladdierer. Wir haben ihn bereits in einem früheren Kursabschnitt aus zwei Halbaddierern aufgebaut, die mit AND-, OR- und NOT-Gattern realisiert wurden. Die Schaltung eines Halbaddierers kann durch Verwendung eines XOR-Gatters vereinfacht werden:



Zwei dieser Halbaddierer werden zu einem Volladdierer zusammengeschaltet:



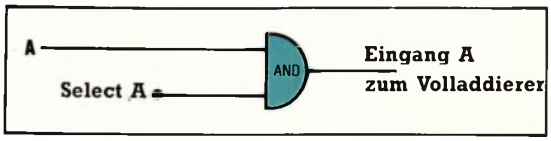
Damit der Volladdierer alle ALU-Funktionen ausführen kann, braucht er einige Zusatzschaltungen, die besondere Steuersignale einspeisen. Das wichtigste Steuersignal ist das „mode select"-Signal (Wahl der Betriebsart). Der Eingang „Übertrag von der vorigen Stelle" wird ausschließlich bei arithmetischen, nicht aber bei Logik-Operationen benötigt. Das „mode select"-Signal schaltet den Übertrags-Eingang dann über ein AND-Gatter ein oder aus:

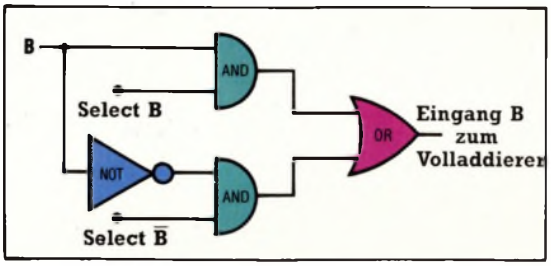


Bei arithmetischen Funktionen steht das „mode select"-Signal auf Eins, und der Übertrag wird durch das Gatter übertragen. Wenn er nicht gebraucht wird, steht „mode select" auf Null. Auf ähnliche Weise werden auch die anderen beiden Eingänge mit AND-Gattern versehen, um die Einspeisung von Bit A, Bit B oder beider Bits gleichzeitig zu steuern.

Für die Subtraktion durch Addition des Zwei-

erkomplements einer Zahl ist die Berechnung des Komplements erforderlich – alle Einsen müssen in Nullen, und alle Nullen in Einsen verwandelt werden. Zur Subtraktion wird also die Zugriffsmöglichkeit auf die Negation von Bit B gebraucht. Zu diesem Zweck wird Eingang B durch ein NOT-Gatter geschleust. Das dazugehörige „select"-Signal wird über ein weiteres AND-Gatter geliefert. Hier die gesamte Schaltung für die Eingabe von A, B mit den Steuersignalen:



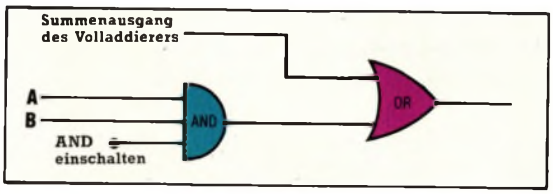


Durch Verwendung von vier Steuersignalen können alle arithmetischen Funktionen ausgeführt werden. Die Tafel gibt die jeweilige Kombinationen der Steuersignale an:

Funktion	Mode Select	Select A	Select B	Select B̄
Addition	1	1	1	0
Subtraktion	1	1	0	1
A inkrementieren (Setzt 1. Übertrags-Eingang auf 1)	1	1	0	0
B inkrementieren	1	0	1	0

Bei Logikfunktionen ist der Übertragseingang unnötig – „mode select" auf Null schaltet ihn ab. Die logische XOR-Funktion ergibt sich jetzt am Summenausgang mit einem HI (Eins) auf Select A und Select B und einem LO (Null) als „mode select"-Signal.

Zur Realisierung der AND-Funktion sind getrennte Eingänge für A und B sowie ein besonderes AND-Select-Signal nötig:



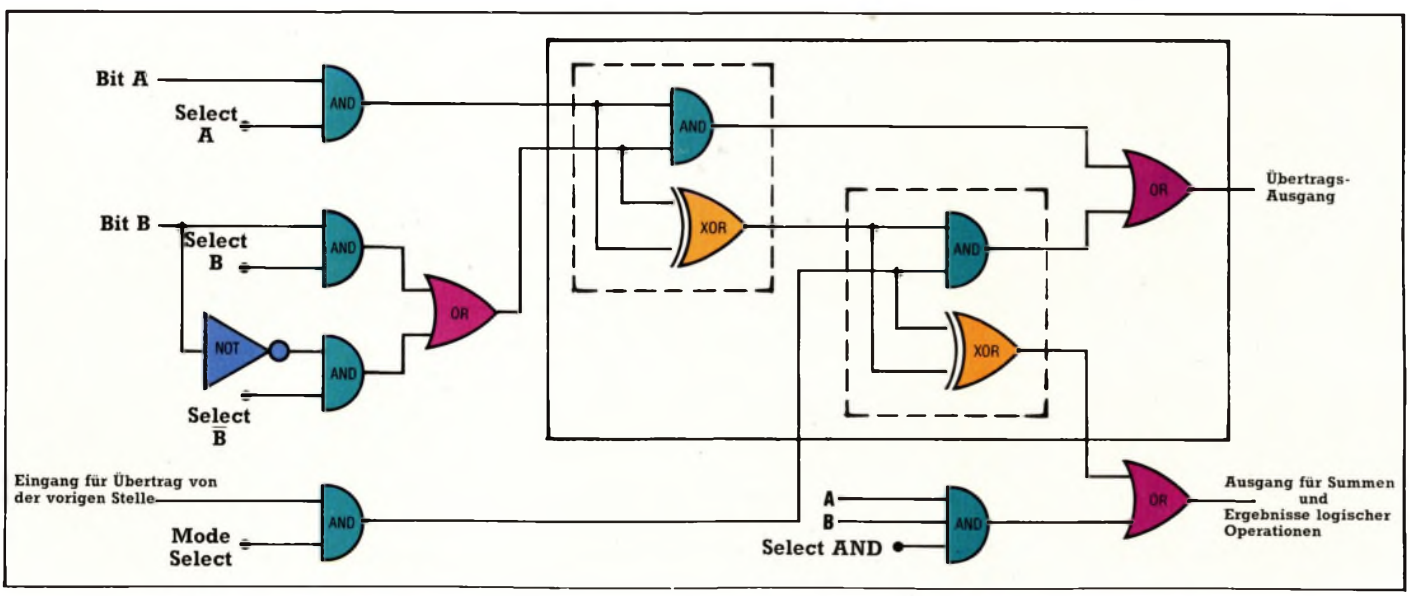
Als letztes fehlt noch die OR-Funktion. Dafür werden XOR- und AND-Ausgänge durch ein OR-Gatter verknüpft, wobei sich diese Wahrheitstafel ergibt:

A	B	Ausgabe	Funktion
0	0	0	
0	1	1	XOR-Funktion
1	0	1	
1	1	1	AND-Funktion

Die folgende Zusammenstellung zeigt, welche Kombination von Steuersignalen zu den einzelnen Logikfunktionen gehört:

Funktion	Mode Select	Select A	Select B	Select B̄	Select AND
XOR	0	1	1	0	0
AND	0	0	0	0	1
OR	0	1	1	0	1

Unten auf dieser Seite ist die vollständige Schaltung der Ein-Bit-ALU mit dem zentralen Volladdierer und den Ergänzungen für die Steuersignale abgebildet. Zur Parallelverarbeitung von acht Bits sind acht dieser Schaltungen nötig. Der Übertragsausgang des achten Bits dient als Flag-Register des Prozessor-Status-Registers.

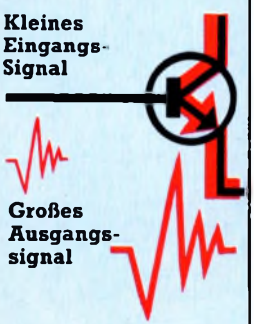


Transistor-Logik

In diesem Kursabschnitt werden einfache Logik-Schaltungen mit Transistoren aufgebaut.

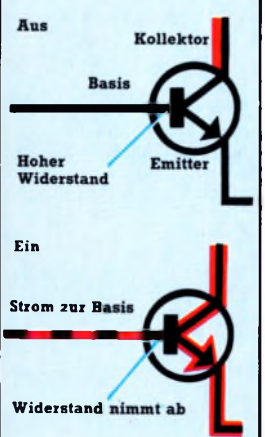
Transistor als Verstärker

An jedem Transistor finden sich drei Anschlußdrähte, die mit den im Gehäuse verborgenen Elementen des Bauteils (Basis, Emitter und Kollektor) verbunden sind. Wenn die an der Basis angelegte Spannung wechselt, ändert sich der Widerstand der Kollektor-Emitter-Strecke im gleichen Takt. Mit einem vergleichsweise geringen Strom zur Basis lassen sich sehr große Ströme zwischen Kollektor und Emitter regeln – der Transistor verstärkt den kleinen Basis-Strom.



Transistor als Schalter

Bei einem bestimmten Sättigungspunkt führt auch ein weiteres Ansteigen des Basis-Stromes nicht mehr zu einem vergrößerten Fluß vom Kollektor zum Emitter. Ist der Basisstrom hingegen sehr klein, „sperrt" der Transistor, das heißt, der Widerstand zwischen den beiden anderen Anschlüssen wird groß.



Transistoren werden für zwei ganz unterschiedliche Funktionen gebraucht: Sie dienen entweder als Verstärker für ein elektrisches Signal oder zum Ein- und Ausschalten eines elektrischen Stromes. In Computern setzt man sie vorwiegend als Schalter ein. Eine Transistorschaltung läßt sich so aufbauen, daß sie bestimmte Muster von Ein- und Ausschaltzuständen speichert. Der Computer behandelt ein solches Muster wie eine Dualzahl. Logische „Gatter" sind andersartige Schaltkreise, mit denen sich zwei bestimmte Ein/Aus-Muster beispielsweise addieren lassen.

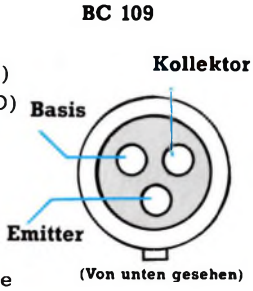
Beim Zusammenbau der auf der nächsten Seite beschriebenen Logikschaltungen wird Ihnen auffallen, daß ein aus Einzel-Transistoren hergestellter Computer nicht nur sehr groß, sondern auch extrem teuer wäre. Das war früher tatsächlich so – Computer zu heutigen Preisen wurden erst durch die Entwicklung von „ICs" möglich. Ein Integrated Circuit besteht aus Hunderten von Transistoren, die gemeinsam auf einem winzigen, plastikummantelten Silizium-Chip sitzen. Für die meisten Anwendungen kann man die Chips in Standardausführung (TTL) einsetzen. Vier AND-Gatter in einem IC-Gehäuse kosten nur wenige Groschen. Die Chips für schwierigere Aufgaben, etwa Binärzähler, sind etwas teurer: Ab 2 Mark sind Sie dabei.

Die Methode, immer mehr Schaltkreise auf einem einzigen Chip zu vereinen, führt zum VLSI-Bauteil (Very Large Scale Integration). Damit sind sehr komplexe Rechen- und Steueroperationen möglich. VLSI-Chips bestehen aus Tausenden von Schaltungen in einem vergleichsweise winzigen Gehäuse. Der Microprozessor Ihres Computers ist ebenso ein VLSI-Chip wie die ICs für die Bildschirmsteuerung, Tonerzeugung und die Schnittstellen-Controller. Das Arbeitsprinzip dieser Chips ist jedoch grundsätzlich das gleiche wie bei den drei nebenstehenden Logik-Schaltungen – nur die Anzahl der Schaltkreise und die Pakkungsdichte sind tausendfach höher.

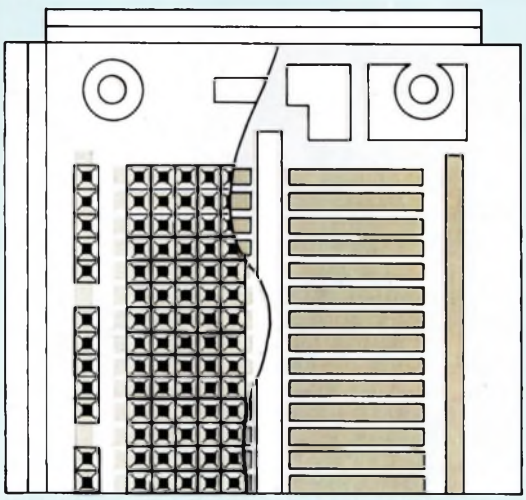
Es gibt verschiedene Techniken, VLSI-Chips herzustellen. Sie variieren daher stark – sowohl im Preis als auch in der Arbeitsgeschwindigkeit und beim Stromverbrauch. In stationären Computern finden sich meist MOS-Chips (Metall-Oxid-Silizium), während man für batteriebetriebene Geräte CMOS-Typen (Complementary Metall-Oxid-Silizium) verwendet. Sie arbeiten langsamer, aber energiesparend.

Das benötigen Sie:

1 Experimentier-Stecksockel
(z. B. Experimentor 300)
2 Transistoren BC 109
2 rote Leuchtdioden (LED)
1 grüne Leuchtdiode (LED)
3 Widerstände 500 Ohm
2 Widerstände 15 kOhm
2 Taster (Schließer)
1 Trockenbatterie, 9 Volt
1 Batterie-Anschlußclip
ein paar kurze Drahtstücke



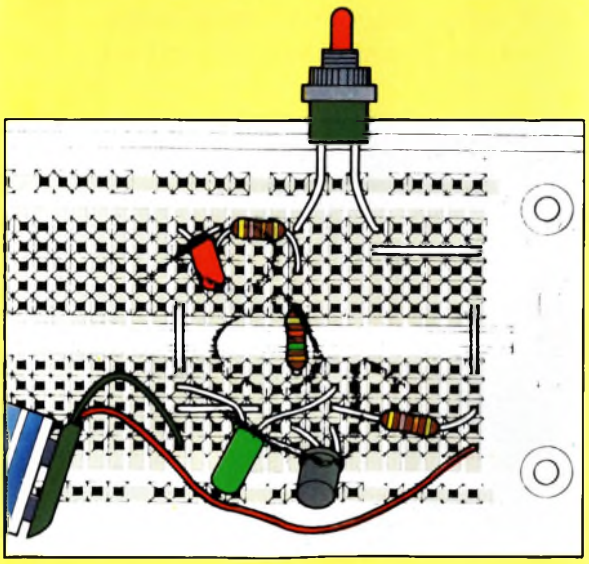
Experimentier-Stecksockel

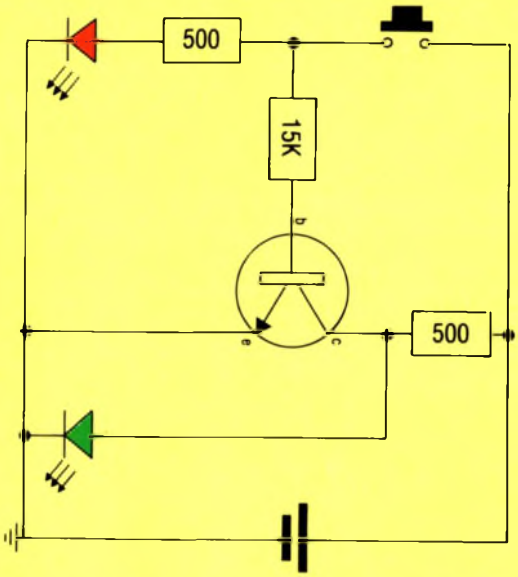


Mit „Breadboards" können Sie eine Schaltung sehr schnell und ohne zeitraubende Lötarbeiten aufbauen. Dabei handelt es sich um wiederverwendbare Platinen, in die Bauteile nur eingesteckt werden. Die Anschlußdrähte werden von metallischen Krallen festgehalten. Jeweils fünf Anschlußpunkte sind miteinander elektrisch verbunden. Durch diese Aufteilung in Felder ist es recht einfach, eine Schaltung auf der Platine zu arrangieren. Mit kurzen Drahtstücken lassen sich die unterschiedlichen Anschlußfelder verbinden. Wählen Sie keinen zu kleinen Experimentiersockel!

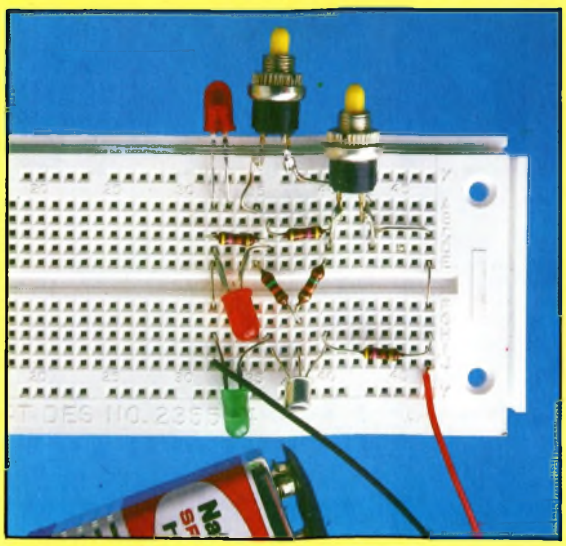
Aufbau logischer Gatter

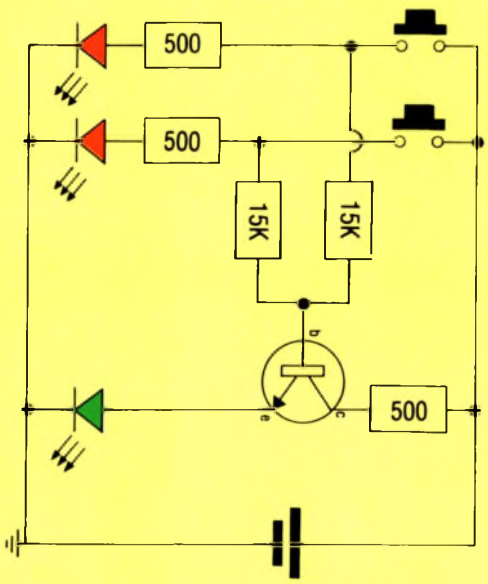
Die hier abgebildeten Baupläne zeigen Schalt-Transistoren in „logischen Gattern". Sie können die Schaltungen nacheinander aufbauen und dabei immer dieselben Bauteile und einen Experimentiersockel verwenden. Die Funktionen der hier vorgestellten Schaltungen sind „solid state" (fest verdrahtet). Sie lassen sich nur durch eine andere Leitungsführung abwandeln. Zur Eingabe in die logischen Gatter dienen Taster mit Leuchtdioden (LEDs). Anders in einem Computer: Hier ist der Eingang eines logischen Gatters meist mit dem Ausgang eines anderen verbunden. Wenn Sie mit den Gattern vertraut geworden sind, bekommen sie vielleicht Lust, einmal etwas Komplizierteres herzustellen – „füttern" Sie doch einmal ein Gatter mit dem Ausgangssignal eines anderen!



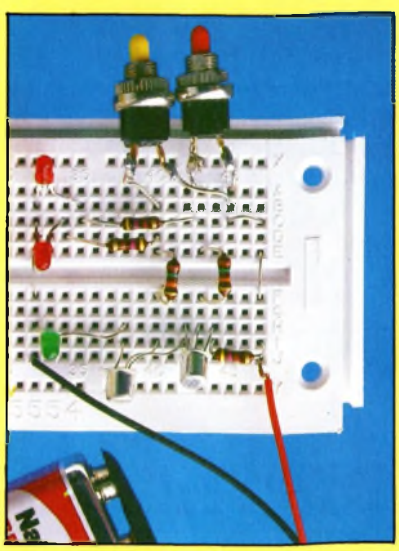


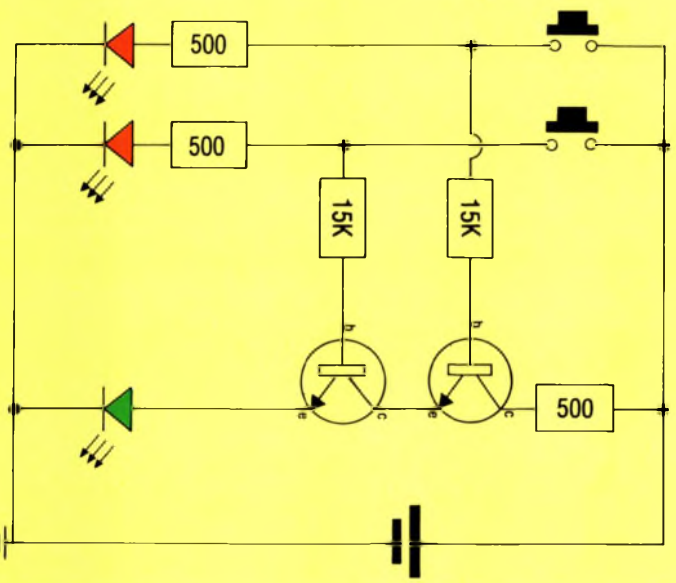
NOT-Gatter
Dies ist das einfachste aller logischen Gatter. Es hat nur einen einzigen Eingang (rote LED und Taster) und auch nur einen Ausgang (grüne LED). Bei geöffnetem Taster fließt kein Strom zur Basis des Transistors – also bleibt der Widerstand zwischen Kollektor und Emitter hoch, und der Strom nimmt seinen Weg durch die grüne LED. Ist also der Taster nicht gedrückt (Eingabe 0), leuchtet die grüne Diode. Bei gedrücktem Taster fließt ein Strom zur Basis des Transistors, der dadurch leitend wird (Widerstand zwischen Kollektor und Emitter niedrig). Jetzt nimmt der Strom seinen Weg durch den Transistor hindurch, ohne die LED zu passieren (leuchtet nicht).





OR-Gatter
Die Schaltung für das NOT-Gatter läßt sich recht einfach zu einem OR-Gatter umbauen. Zuerst wird die Ausgangs-LED so plaziert, daß sie von einem Strom durch den Transistor angesteuert wird. Das OR-Gatter hat zwei Eingänge, beide mit Taster und LED. Ist einer der beiden Taster gedrückt, fließt ein Strom zur Basis des Transistors. Er wird leitend, und die Diode strahlt Licht aus.

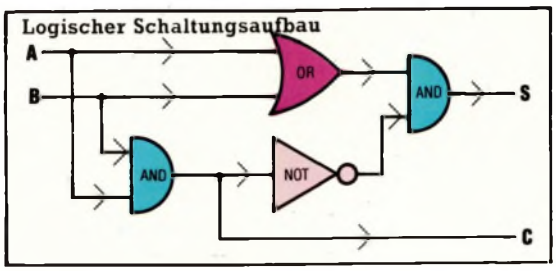




AND-Gatter
Für ein AND-Gatter werden zwei Transistoren hintereinander in die Ausgangsleitung geschaltet. Jeder Transistor bekommt aber einen eigenen Eingangs-Taster. Nur wenn beide Taster gedrückt sind (also beide Eingänge 1 sind), kann der Strom nacheinander die beiden Transistoren passieren und die Ausgangs-LED zum Leuchten bringen.

Halbaddierer mit Logik-Gattern

In dieser Folge unseres Bastelkurses wollen wir über die AND-, OR- und NOR-Gatter hinausgehen und mit zwei integrierten Schaltkreisen einen Halbaddierer aufbauen.



Die einzelnen Logik-Gatter aus dem letzten Kursteil stellen die Grundlage komplexerer Digital-Schaltkreise dar. Mit einem Halbaddierer lassen sich zwei Einzel-Bits addieren. Er stellt zwei Eingänge für die zu addierenden Bits sowie zwei Ausgänge bereit: einen Summen- und einen Übertragsausgang. Mit einer Wahrheitstabelle läßt sich das Verhalten des Halbaddierers so darstellen:

A(=1.Bit)	B(=2.Bit)	S(=Summe)	Ü(=Übertrag)
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

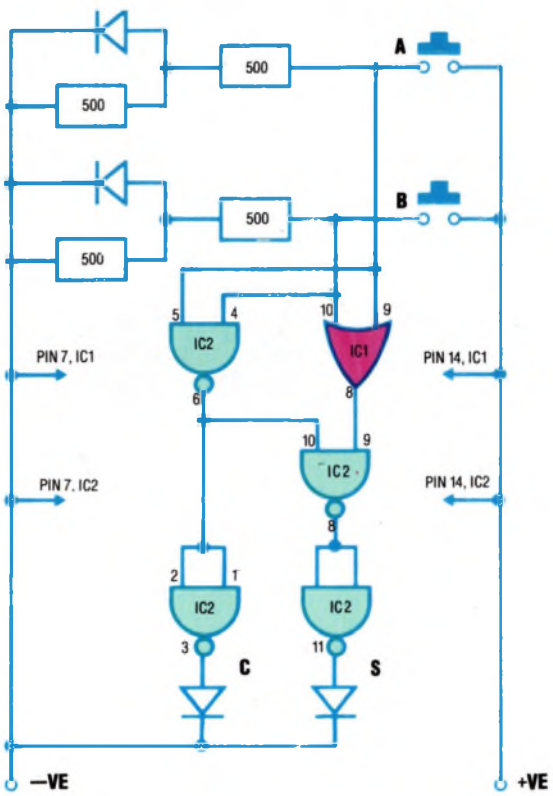
Der Ausgang S zeigt immer die Summe der beiden Eingangsbits. Sind beide Bits 1, ergibt sich im binären System 10, für deren Darstellung ein zusätzlicher Ausgang Ü (Übertrag) nötig ist. Ausgang S zeigt bei 10 also 0, Ausgang Ü die 1.

Wenn (in einem Acht-Bit-Rechner) zwei Acht-Bit-Worte addiert werden sollen, kommt man mit einem einzigen Halbaddierer nicht weit. Erst 16 Halbaddierer zusammen vollbringen dieses Kunststück: Die beiden ersten Bits werden vom ersten Halbaddierer zusammengezählt, der am Summenausgang die letzte Stelle des Ergebnisses zeigt. Der jeweilige Übertrag wird zum Additionsergebnis der nächsten zwei Bits hinzugezählt, der Übertrag dieser Addition wird zur Summe des dritten Bits addiert – und so weiter.

Auch ein ganz primitiver Halbaddierer würde aus mindestens zehn Transistoren bestehen, wenn er aus den uns bekannten Gatterschaltungen aufgebaut wäre. Zum Glück gibt es AND, NAND, OR, NOR und andere logische Gatter als IC. Mit jeweils vier Gattern pro Bauteil wird somit die Schaltung nicht nur einfacher, sondern auch preisgünstiger.

Einen einfachen, aus OR-, AND- und NOT-Gattern aufgebauten Addierer zeigt der abgebildete Schaltplan. Da die hier verwendeten ICs nur mit jeweils gleichen Gattern erhältlich sind, haben wir die Schaltung durch Verwendung möglichst weniger Gatter-Typen vereinfacht: Es sind nur vier NAND- und ein einziges

Elektrischer Schaltungsaufbau



OR-Gatter nötig; daher brauchen Sie insgesamt nur zwei ICs.

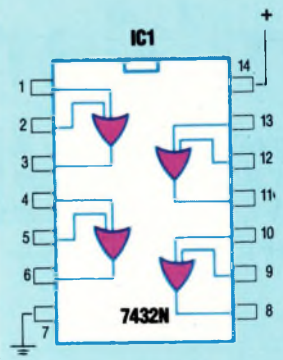
Sie werden wahrscheinlich bei der Montage feststellen, daß für die vergleichsweise geringe Leistung dieser Konstruktion doch recht viel Mühe nötig ist. Es geht zwar sehr viel schneller als mit Einzel-Bauteilen – aber soll man einen kompletten Computer auf diese Art herstellen?

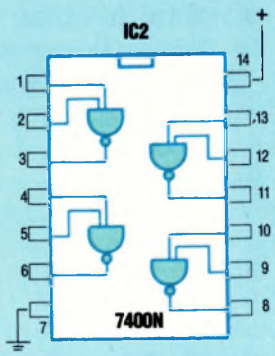
In der Praxis werden ICs nur selten in dieser Weise verwendet – viel häufiger kommt es vor, daß sie im Computer für einen kleinen „Aushilfs-Job" abkommandiert werden. Die komplexeren Aufgaben sind den größeren Chips mit mehr Ein- und Ausgängen vorbehalten. Damit lassen sich dann etwa zwei 4-Bit-Zahlen addieren.

Das brauchen Sie:

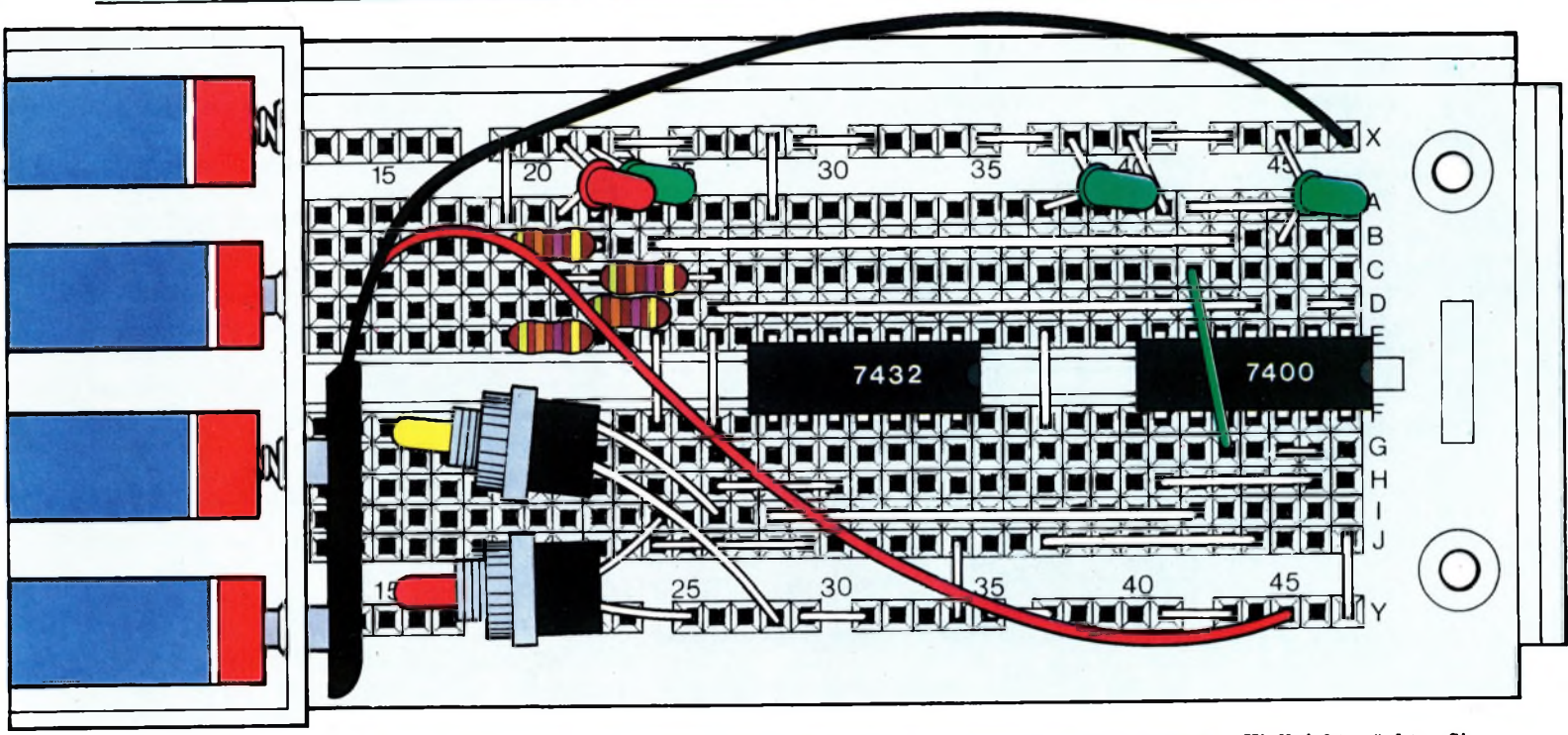
4 Widerstände, 500 Ohm, 0,25 Watt
4 Leuchtdioden (LEDs)
2 Taster (Schließer)
1 IC 7400N
1 IC 7432N
4 Baby- oder Monozellen 1,5 Volt
1 Batteriehalterung
1 Batterieclip
1 Experimentier-Stecksockel
einige kurze Drahtstücke

Anschlußbelegung

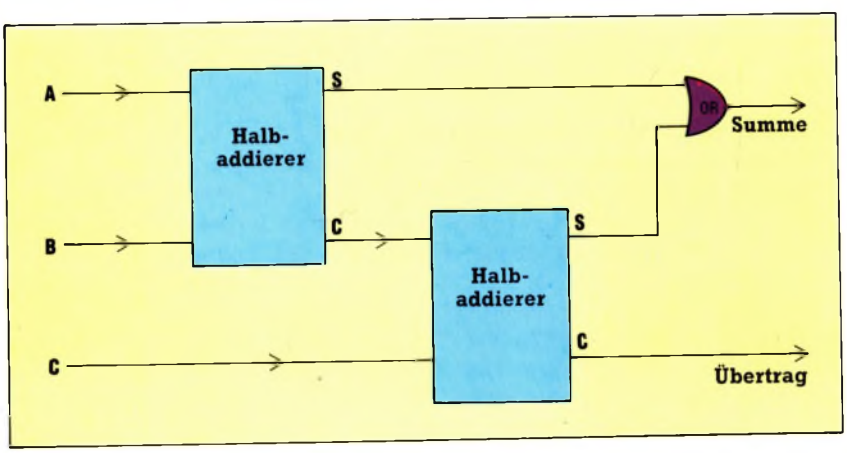




Bei den verwendeten ICs handelt es sich um 14-Pin-TTL-Bauteile. Der 7400N enthält vier NAND-Gatter, der 7432N stellt vier OR-Gatter zur Verfügung. Auf der Zeichnung können Sie erkennen, wie die Gatteranschlüsse liegen. Sie brauchen nur noch für die richtige Verdrahtung zu sorgen. Auf einer Seite des Chips ist ein Ausschnitt, der zeigt, wie das Bauteil eingesetzt werden muß. Denken Sie auch an die Versorgungsspannung, die über Pin 7 und 14 eingespeist wird.



Nach dem Entwurf der Schaltung muß man sich Gedanken über die Anordnung der Bauteile auf der Platine machen. Für die Konzeption können Sie spezielle Vordrucke verwenden, die Fotokopie einer leeren Platine erfüllt jedoch den gleichen Zweck. Am einfachsten wird es, wenn der reale Schaltungsaufbau dem Plan ähnelt – man irrt sich nicht so leicht bei der Verdrahtung. Wenn Sie sich genau an das Original halten, kann nichts schiefgehen – alle Bauteile sitzen an der richtigen Stelle.



Vielleicht möchten Sie zur Übung Ihren Halbaddierer zu einem Volladdierer aufrüsten. Mit der hier abgebildeten Schaltung werden nicht einfach nur zwei Bits addiert, sondern auch der Übertrag vom vorhergehenden Bit ist berücksichtigt. Eine Gruppe von Volladdierern kann komplette Bit-Worte zusammenrechnen. Im einfachsten Fall werden dazu zwei Halbaddierer wie hier im Bild zusammengeschaltet. Dabei gelangt das Summensignal des ersten Halbaddierers auf einen Eingang des nächsten Addierers.

Wiederholung

Bevor es im elektronischen Selbstbau-Kursus weitergeht, sollten Sie sich die Zeit für einen kleinen Rückblick auf die behandelten Themenkreise nehmen.

Am Anfang des Kurses kam die Bedeutung geeigneter Werkzeuge zur Sprache. Für Verbesserungen und Reparaturen sollte man dabei unbedingt eine gewisse Minimal-Ausrüstung bereithalten. Zwar läßt sich eine Kreuzschlitzschraube auch mit einem flachen Schraubendreher, eine sehr kleine Mutter auch mit der Pinzette lösen – ohne Schaden für Gerät oder Material geht es so aber auf die Dauer nicht. Auf lange Sicht lohnt die Anschaffung einiger Spezial-Werkzeuge.

Ähnlich ist es auch beim Verdrahten – bei einer mit geeignetem Werkzeug sauber gelöteten Verbindung ist das Fehlerrisiko geringer als bei einer hastigen „Pfuscharbeit". Lötzinn sollte zum Beispiel immer nur auf den Draht, nie auf die Lötkolbenspitze gegeben werden. Die Spitze wird vom Flußmittel des Zinns leicht angegriffen. Bei zwei zu verbindenden Drähten sollten Sie darauf achten, daß die gesamte Kontaktstelle mit Zinn abgedeckt ist. Erst dann nimmt man den Lötkolben beiseite und kühlt durch leichtes Anpusten. Dieses Verfahren verhindert das Entstehen von „kalten" Lötstellen.

Die Grundlagen der digitalen Elektronik haben wir bereits erläutert. Dabei wurden die Funktionen der wichtigsten Bauelemente erklärt, von denen der Widerstand das unkomplizierteste ist.

Kondensatoren dienen im Computer zum Ausfiltern unerwünschter Signalanteile. Jedes Signal wird bei der Übertragung, Verstärkung oder Veränderung mehr oder weniger verzerrt – der Kondensator hilft, das Signal wieder zu entzerren. Erst dadurch kann ein ungestörter Signalfluß erreicht werden.

Trotz der nützlichen Funktionen von Widerständen und Kondensatoren stand der Transistor in der Mitte unserer Bauteil-Kunde. Er hat im Computer die Aufgabe, Signale zu verarbeiten. Je nach Betriebsart arbeitet der Transistor als Verstärker oder als Schalter. Wichtig ist, daß ein Transistor jedes Signal in Abhängigkeit von anderen Signalen ein- oder ausschalten kann. Damit erhält er die Funktion eines logischen Gatters.

Im Selbstbau-Kurs haben Sie auch gesehen, daß man nur wenige Bauteile neben dem Transistor braucht, um die einfachsten Logikschaltungen (NOT-, OR-, AND-Gatter) aufzubauen.

Für ein einzelnes Logik-Gatter gibt es zwar kaum eine sinnvolle Einsatzmöglichkeit, mehrere zusammen können aber Daten ver- und bearbeiten. Im letzten Kursabschnitt haben wir mit den logischen Gattern von zwei ICs einen Halbaddierer hergestellt – eine Schaltung, die zwei Binär-Bits addieren kann.

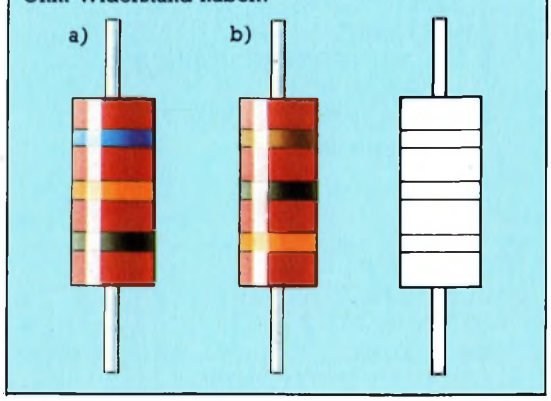
Integrierte Schaltkreise (ICs) sind die kompliziertesten Bauteile der Elektronik. Die von uns bisher verwendeten Transistor-Transistor-Logik-ICs (TTL) gehören zu den SSI-(= Small Scale Integration) Elementen, die auf einem Chip nur relativ wenige Transistoren vereinigen. Früher wurden noch ganze Computer aus SSI-Chips aufgebaut. Heute hat die Anzahl der Transistoren auf einem einzigen Chip gewaltig zugenommen: Schon bei MSI-(Medium Scale Integration) Elementen findet man komplette Logikschaltungen in einem IC.

Inzwischen ist die Technik bei LSI- und sogar VLSI-Chips angelangt (Large bzw. Very Large Scale Integration). Damit läßt sich ein kompletter Prozessor auf einem Chip unterbringen – eine Schaltung aus Tausenden von Einzeltransistoren, deren logische Verknüpfung kaum noch vorstellbar ist. Solche Bauteile sind extrem leistungsfähig und lassen sich dadurch vielfältig einsetzen.

Nachdem wir die Grundlagen kennen, dürfen wir uns an komplexere Aufgaben herantrauen, insbesondere an die Konstruktion nützlicher Zusatzschaltungen für den Heimcomputer. Überprüfen Sie mit den Beispielen auf der nächsten Seite noch einmal Ihre Kenntnisse. Und scheuen Sie sich nicht, in Zweifelsfällen noch einmal nachzuschlagen!

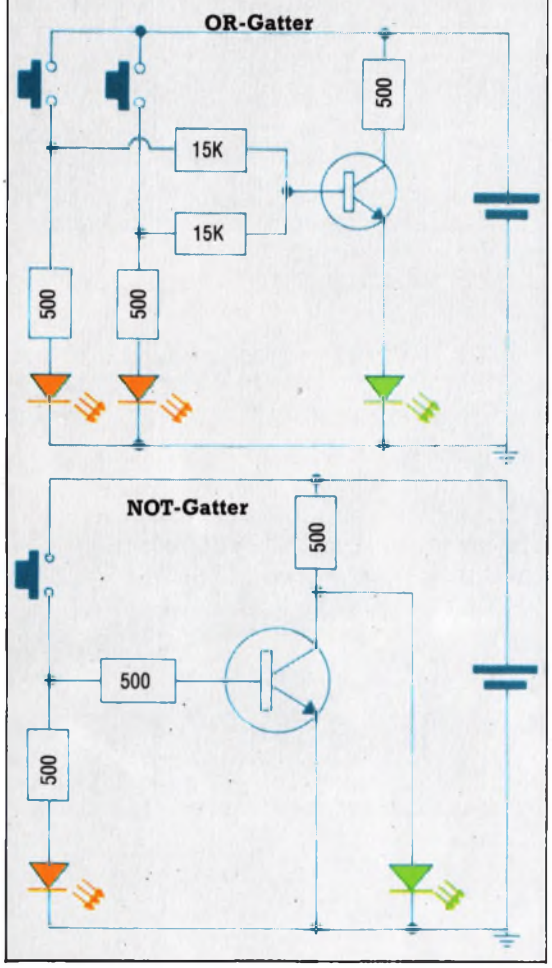
1) Widerstände

An den farbigen Ringen auf einem Widerstand können Sie seinen Wert ablesen. Wieviel Ohm haben die beiden hier abgebildeten Widerstände? Welche Farben würden die Ringe auf einem 150-Ohm-Widerstand haben?



2) NOR-Gatter

An früherer Stelle haben wir NOT-, OR- und AND-Gatter mit Transistoren aufgebaut. In dieser ersten Übung soll auf ähnliche Weise ein NOR-Gatter (NOT+OR) hergestellt werden. Als kleine Hilfe sind noch einmal die Schaltungen des OR- und des NOT-Gatters unten abgedruckt. Das Problem kann auf zwei verschiedene Arten gelöst werden. Die eine Lösung ist, ein OR- und ein NOT-Gatter zu kombinieren.



3) Binär-Dezimal-Wandler

Versuchen Sie, eine Schaltung zu bauen, die Dezimalziffern in Binärzahlen umwandelt. Damit es nicht zu schwierig wird, beschränken wir uns auf zweistellige Binärzahlen, also die Dezimalzahlen von 0 bis 3. Ihre Schaltung sollte vier Eingangstaster haben, die mit 0, 1, 2 und 3 bezeichnet sind. Beim Drücken eines Schalters sollten die entsprechenden binären Bits durch das Aufleuchten von LEDs angezeigt werden. Die Wahrheitstabelle für den Wandler sieht so aus:

NULL Taster	EINS Taster	ZWEI Taster	DREI Taster	Hohes Bit	Niedriges Bit
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

4) BCD oder nicht?

Der BCD-Code (Binär Codiertes Dezimalsystem) steht zwischen den reinen Binärzahlen und dem Dezimalsystem – jede einzelne Dezimalziffer wird in ihr binäres Äquivalent umgewandelt. Eine BCD-Zahl setzt sich dementsprechend aus Vier-Bit-Gruppen zusammen. Die Zahl 53 wird zu 01010011, wobei 0101 für die 5 und 0011 für die 3 steht. Jede zulässige BCD-Ziffer ist also eine Bit-Gruppe zwischen 0000 (dezimal 0) und 1001 (dezimal 9). Bit-Gruppen zwischen 1010 und 1111 sind im BCD-Code nicht zugelassen.

Bauen Sie eine Schaltung auf, die eine Eingabe auf BCD-Gültigkeit prüft. Die Schaltung braucht vier Eingangstaster von B0 bis B3. Damit wird die BCD-Zahl eingegeben. Zwei Ausgänge (eine grüne und eine rote LED) zeigen an, ob die Zahl dem BCD-Code entspricht (grün) oder nicht (rot). Hier die Wahrheitstabelle:

Entsprechende Dezimalzahl	B3	B2	B1	B0	Im BCD-Code gültig	Im BCD-Code ungültig
0	0	0	0	0	1	0
1	0	0	0	1	1	0
2	0	0	1	0	1	0
3	0	0	1	1	1	0
4	0	1	0	0	1	0
5	0	1	0	1	1	0
6	0	1	1	0	1	0
7	0	1	1	1	1	0
8	1	0	0	0	1	0
9	1	0	0	1	1	0
10	1	0	1	0	0	1
11	1	0	1	1	0	1
12	1	1	0	0	0	1
13	1	1	0	1	0	1
14	1	1	1	0	0	1
15	1	1	1	1	0	1

Aus dieser Tabelle können Sie ersehen, daß ein Signal für gültige BCD-Zahlen aus $\overline{B3}+\overline{B2.B1}$ entsteht. Das Signal für eine ungültige Eingabe ergibt sich aus dem NOT von:

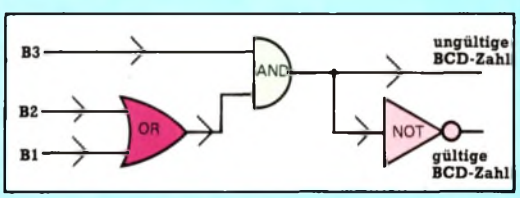
$$\overline{\overline{B3} + \overline{B2.B1}}$$

das sich so vereinfachen läßt:

$$\overline{\overline{B3}} \cdot \overline{\overline{B2.B1}}$$

$$B3 \cdot (B2 + B1)$$

Die Schaltung zur Prüfung auf BCD-Gültigkeit ist recht einfach. Es werden nur drei Eingänge benutzt. Unten ein Vorschlag für den Logik-Aufbau des BCD-Testers:



Lösungen finden Sie im nächsten Abschnitt des Selbstbau-Kurses.

Lösungen

Die Übungsaufgaben aus dem letzten Teil des Kurses werden erklärt.

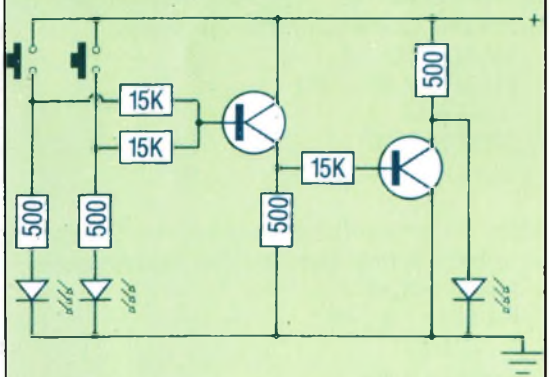
Wie gut kennen Sie sich schon in der Elektronik aus? Die hier von uns vorgeschlagenen Lösungen sind vielleicht anders als die von Ihnen gefundenen. Kein Grund zur Besorgnis – auch in diesem Fall können verschiedene Wege zum gleichen Ziel führen. Entscheidend ist nur, daß sich die Logik der von Ihnen gebauten Schaltung mit den Vorgaben der Wahrheits-Tabelle deckt!

1) Widerstände

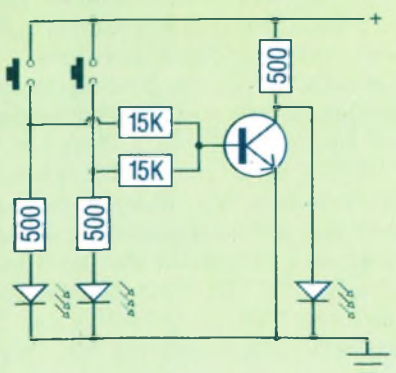
Die Werte der abgebildeten Widerstände sind: a) 6400 Kilo-Ohm und b) 150 Kilo-Ohm. Ein Widerstand von 150 Ohm wäre mit braun-grün-braunen Ringen versehen (In Richtung zum silbernen oder goldenen Ring hin gelesen).

2) NOR-Gatter

Der einfachste Weg zu einem NOR-Gatter ist die Verbindung aus einer OR- und einer NOT-Schaltung wie hier abgebildet:

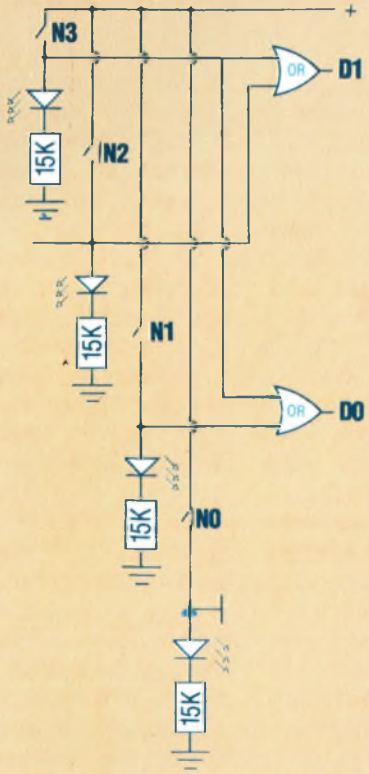


Noch einfacher geht es mit diesem Trick. Dazu wird – ähnlich wie im NOT-Gatter – das Ausgangssignal des Kollektors anstelle des Emittersignals verwendet.



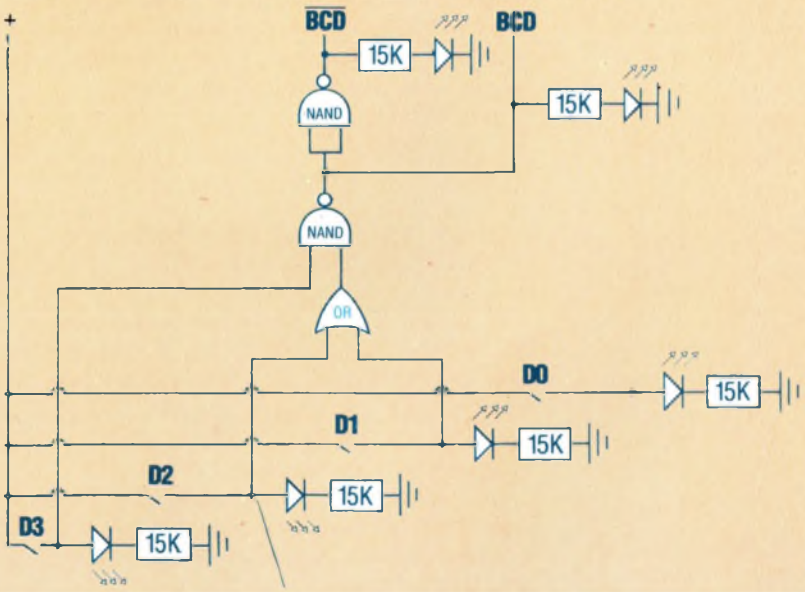
3) Dezimal-Binärwandler

Dafür wird nur ein einziges IC gebraucht – ein TTL-Chip mit vier OR-Gattern.



4) BCD oder nicht?

Die gültigen BCD-Ziffern reichen von 0000 bis 1001, alle höheren Codes sind ungültig. Bei unserer Lösung sieht der BCD-Prüfer so aus:



Keine Angst vor Bits und Bytes!

Der Computer rechnet nur mit zwei Ziffern, mit 0 und 1.

Die Worte Bit und Byte werden immer dann auftauchen, wenn über Computer gesprochen wird. Der Begriff Bit kommt aus dem Englischen und bedeutet „BInary DigiT" und heißt soviel wie Binärziffer. Ein Byte ist eine Gruppe von acht Bits. Das Wort wird von dem englischen Begriff „By eight" abgeleitet.

Mit diesen Bits und Bytes arbeitet also ein Computer. Er rechnet nicht wie wir im gewohnten Zehnersystem, in dem alle Zahlen unter Verwendung der Ziffern 0 bis 9 dargestellt werden, sondern im Binärsystem.

Nur in einigen wenigen Fällen müssen wir umdenken und auch im Binärsystem rechnen. Dies scheint recht schwierig zu sein, ist aber, wie beim Zehnersystem, eine reine Gewöhnungssache.

Doch zurück zum Bit. Klären wir, was der Computer mit den Bits und den Bytes macht.

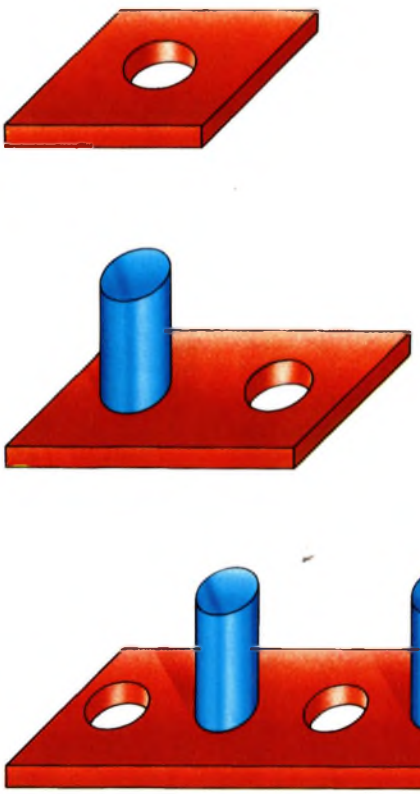
Ein Bit ist also die kleinste Informationseinheit, die ein Computer verarbeiten kann. Mit dem Bit kann der Computer nur eine Null oder eine Eins darstellen. Mit einer Gruppe von acht Bits, also einem Byte, kann ein Computer nun schon Buchstaben und Zahlen darstellen. Doch was spielt sich im Rechner ab?

„Ein" oder „Aus"

Computer sind elektronische Maschinen, die mit elektrischen Signalen arbeiten. Ein elektrisches Signal kann entweder ein „Ein" oder ein „Aus" sein. Besser veranschaulicht wird dies an dem abgebildeten Beispiel, das ein Brettchen mit Loch zeigt. Obwohl dieses Brettchen nur ein Loch hat, können zwei Ziffern dargestellt werden. Und nach diesem Prinzip arbeitet ein Computer.

Wenn das Loch im Brett leer ist, wird eine „Null", wenn ein 'Dübel' im Loch steckt, wird eine „Eins" dargestellt. Beim Computer wird dies mit elektrischen Signalen gemacht. Ist es „Aus", wird eine Null beschrieben, ist es auf „Ein" geschaltet, bedeutet dies eine Eins.

Nimmt man nun ein Brettchen mit zwei Lö-



Bits und Bytes

Ein Bit stellt die kleinste Informationseinheit dar, die ein Computer verarbeiten kann. Wie auf dieser Abbildung gezeigt, kann ein Holzbrett mit nur einem Loch eine Eins oder eine Null repräsentieren. Der Computer macht genau dasselbe mit elektrischen Signalen, die entweder ein hohes oder ein niedriges Potential haben und so eine 1 oder eine 0 darstellen. Wenn ein Holzbrett zwei Löcher hat, haben wir es mit vier möglichen Kombinationen von Löchern und 'Dübeln' zu tun. Bei einem Brett mit acht Löchern werden 256 verschiedenen Kombinationen möglich. Computer benutzen die Bits in Achtergruppen, die als Bytes bezeichnet werden. Jedes Byte kann eine Zahl zwischen 0 und 255 darstellen.

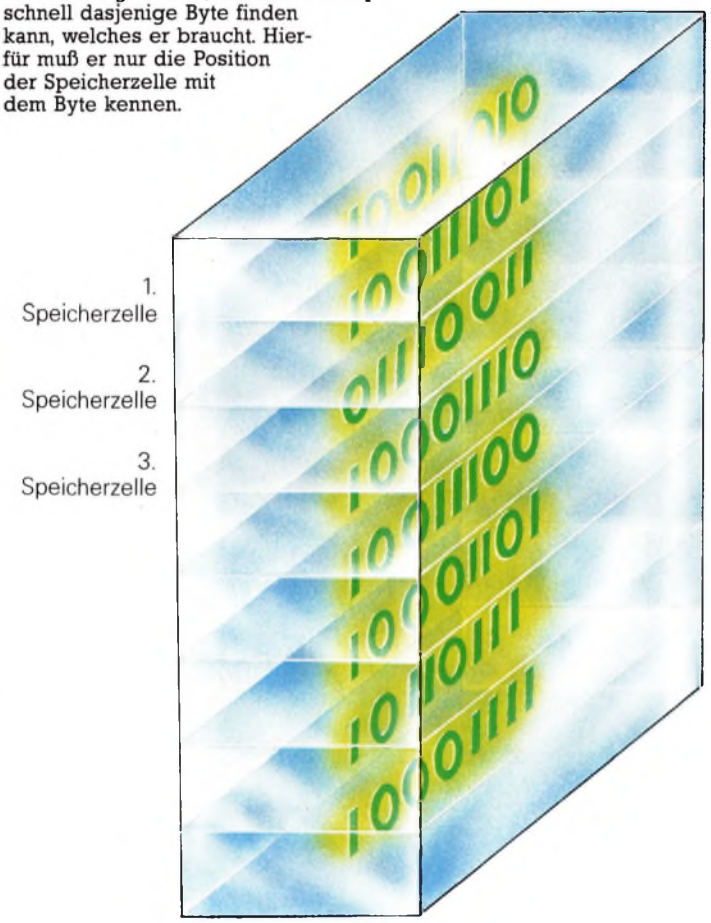
TONY LODGE

Das Bit kann zwei Zustände annehmen: „Ein" oder „Aus". Wie stellt der Computer diese verschiedenen Zustände dar? Er bedient sich verschiedener Stromspannungen. Dabei ist eine Spannung je nach Rechner zwischen 1,6 und 2,4 Volt die Grenze zwischen „Ein" und „Aus". Der Computer benutzt also eine Spannung von beispielsweise 5 Volt für den „Ein"-Zustand und 0 Volt für „Aus".

0	00000000	128	10000000
1	00000001	129	10000001
2	00000010	130	10000010
3	00000011	131	10000011
4	00000100	132	10000100
5	00000101	133	10000101
6	00000110	134	10000110
7	00000111	135	10000111
8	00001000	136	10001000
9	00001001	137	10001001
10	00001010	138	10001010
11	00001011	139	10001011
12	00001100	140	10001100
13	00001101	141	10001101
14	00001110	142	10001110
15	00001111	143	10001111
16	00010000	144	10010000
17	00010001	145	10010001
18	00010010	146	10010010
19	00010011	147	10010011
20	00010100	148	10010100
21	00010101	149	10010101
22	00010110	150	10010110
23	00010111	151	10010111
24	00011000	152	10011000
25	00011001	153	10011001
26	00011010	154	10011010
27	00011011	155	10011011
28	00011100	156	10011100
29	00011101	157	10011101
30	00011110	158	10011110
31	00011111	159	10011111
32	00100000	160	10100000
33	00100001	161	10100001
34	00100010	162	10100010
35	00100011	163	10100011
36	00100100	164	10100100
37	00100101	165	10100101
38	00100110	166	10100110
39	00100111	167	10100111
40	00101000	168	10101000
41	00101001	169	10101001
42	00101010	170	10101010
43	00101011	171	10101011
44	00101100	172	10101100
45	00101101	173	10101101
46	00101110	174	10101110
47	00101111	175	10101111
48	00110000	176	10110000
49	00110001	177	10110001
50	00110010	178	10110010
51	00110011	179	10110011
52	00110100	180	10110100
53	00110101	181	10110101
54	00110110	182	10110110
55	00110111	183	10110111
56	00111000	184	10111000
57	00111001	185	10111001
58	00111010	186	10111010
59	00111011	187	10111011
60	00111100	188	10111100
61	00111101	189	10111101
62	00111110	190	10111110
63	00111111	191	10111111
64	01000000	192	11000000
65	01000001	193	11000001
66	01000010	194	11000010
67	01000011	195	11000011
68	01000100	196	11000100
69	01000101	197	11000101
70	01000110	198	11000110
71	01000111	199	11000111
72	01001000	200	11001000
73	01001001	201	11001001
74	01001010	202	11001010
75	01001011	203	11001011
76	01001100	204	11001100
77	01001101	205	11001101
78	01001110	206	11001110
79	01001111	207	11001111
80	01010000	208	11010000
81	01010001	209	11010001
82	01010010	210	11010010
83	01010011	211	11010011
84	01010100	212	11010100
85	01010101	213	11010101
86	01010110	214	11010110
87	01010111	215	11010111
88	01011000	216	11011000
89	01011001	217	11011001
90	01011010	218	11011010
91	01011011	219	11011011
92	01011100	220	11011100
93	01011101	221	11011101
94	01011110	222	11011110
95	01011111	223	11011111
96	01100000	224	11100000
97	01100001	225	11100001
98	01100010	226	11100010
99	01100011	227	11100011
100	01100100	228	11100100
101	01100101	229	11100101
102	01100110	230	11100110
103	01100111	231	11100111
104	01101000	232	11101000
105	01101001	233	11101001
106	01101010	234	11101010
107	01101011	235	11101011
108	01101100	236	11101100
109	01101101	237	11101101
110	01101110	238	11101110
111	01101111	239	11101111
112	01110000	240	11110000
113	01110001	241	11110001
114	01110010	242	11110010
115	01110011	243	11110011
116	01110100	244	11110100
117	01110101	245	11110101
118	01110110	246	11110110
119	01110111	247	11110111
120	01111000	248	11111000
121	01111001	249	11111001
122	01111010	250	11111010
123	01111011	251	11111011
124	01111100	252	11111100
125	01111101	253	11111101
126	01111110	254	11111110
127	01111111	255	11111111

Bytes im Speicher

Bytes sind Gruppen aus je acht Bits. Jedes Byte wird vom Computer zum Speichern von Zahlen benutzt, die im Bereich von 0 bis 255 liegen können. Jedes Byte wird in einer getrennten Speicherzelle bewahrt. Die Speicherzellen sind systematisch geordnet, damit der Computer schnell dasjenige Byte finden kann, welches er braucht. Hierfür muß er nur die Position der Speicherzelle mit dem Byte kennen.



TONY LODGE

chern, so können vier verschiedene Zustände gezeigt werden. Man zählt von null bis drei. Diese Möglichkeiten ergeben sich so: die beiden Löcher können leer sein; in dem rechten Loch kann ein Dübel stecken, wobei das linke Loch leer ist; in dem linken Loch kann ein Dübel stecken, wobei das rechte Loch leer ist; oder in beiden Löchern stecken Dübel. Auf dem unteren Brettchen mit acht Löchern in der Zeichnung können durch acht Dübel insgesamt 256 Kombinationen gezeigt werden. Diese Gruppe von acht Bits nennt man also ein Byte. Ein einzelnes Byte kann demnach 256 verschiedene Zustände zeigen.

Wenn man nun sagt, der Computer speichert ein Byte, muß man sich also vorstellen, daß der Computer eine Zahl zwischen 0 und 255 in seinem Speicher ablegt. Der Speicher kann mit aufeinandergestapelten Schubladen verglichen werden. In ihnen werden die einzelnen Bytes aufbewahrt. Soll ein bestimmtes Byte gefunden werden, muß der Computer wissen, in welchem Speicher er zu suchen hat.

Alle Zahlen von 0 bis 255 können mit Hilfe eindeutiger Kombinationen von 0 bis 1 dargestellt werden (Tabelle links). Die Computer speichern und benutzen Bits in Achter-Gruppen. Acht Bits zusammengefaßt werden immer als Byte bezeichnet.

Ein Gedächtnis, das nichts vergißt

Wie der Computer Daten, Informationen oder Programme speichert, verwaltet und auf Abruf wieder herausgibt.

Beim Menschen kann man sich das Gedächtnis als großes Lagerhaus vorstellen, in dem Erfahrungen oder Eindrücke gespeichert werden. Bei einem Computer meint das Wort Gedächtnis im Prinzip das gleiche, mit dem Unterschied, daß ein Computergedächtnis wesentlich limitierter arbeitet.

Das Wort Gedächtnis beinhaltet zwei Dinge: Speichern und Erinnern. Denn nur Informationen zu speichern und nicht in der Lage zu sein, die Information wieder abzurufen, ist genauso unsinnig wie Informationen abrufen zu wollen, die überhaupt nicht vorhanden sind.

Das menschliche Gedächtnis teilt sich in Kurzzeit- und Langzeitgedächtnis.

Kurzzeit- und Langzeitgedächtnis

Ein Mann, der die Straße überqueren will, wird am Straßenrand stehenbleiben und solange warten, bis der herankommende Wagen vorbeigefahren ist. Hat dann der Mann die Straße gefahrlos überquert, wird er sich mit Sicherheit nicht mehr an den Wagen erinnern. Die Situation wurde im Kurzzeitgedächtnis gespeichert. Wenn aber im gleichen Wagen zwei maskierte Männer und die Ehefrau des Mannes gesessen hätten, dann könnte sich unser Mann nicht nur an den Wagen, sondern auch an dessen Farbe, Typ und vielleicht sogar an das Kennzeichen erinnern. Dies wäre dann ein Fall für das Langzeitgedächtnis.

Genaugenommen besitzt ein Computer, ebenso wie der Mensch, ein Kurz- und ein Langzeitgedächtnis. Das Langzeitgedächtnis speichert Programme und Informationen ab, die der Benutzer später weiterverwenden will. Cassetten, Disketten oder ROM-Module stellen diesen Gedächtnistyp dar. Das Kurzzeitgedächtnis besteht in dem RAM-Chip, welches nur zeitweilig benutzt wird. In dem Moment, in dem Sie Ihren Computer ausschalten, wird das Kurzzeitgedächtnis, also das RAM, vollständig gelöscht. Alle hier gespeicherten Informationen sind ein für allemal verloren.

Die Analogie mit dem menschlichen Gedächtnis findet jetzt ihr Ende. Wenn ein Computer ein Programm bearbeitet, so liest er das Programm mit allen notwendigen Daten aus seinem Langzeitgedächtnis und speichert es im Kurzzeitgedächtnis, dem RAM, und kann so schnell und direkt auf diese Daten zugreifen. Die Art und Weise, wie ein Computer diese Daten abruft und speichert, ist von der Arbeitsweise des menschlichen Gedächtnisses vollkommen verschieden.

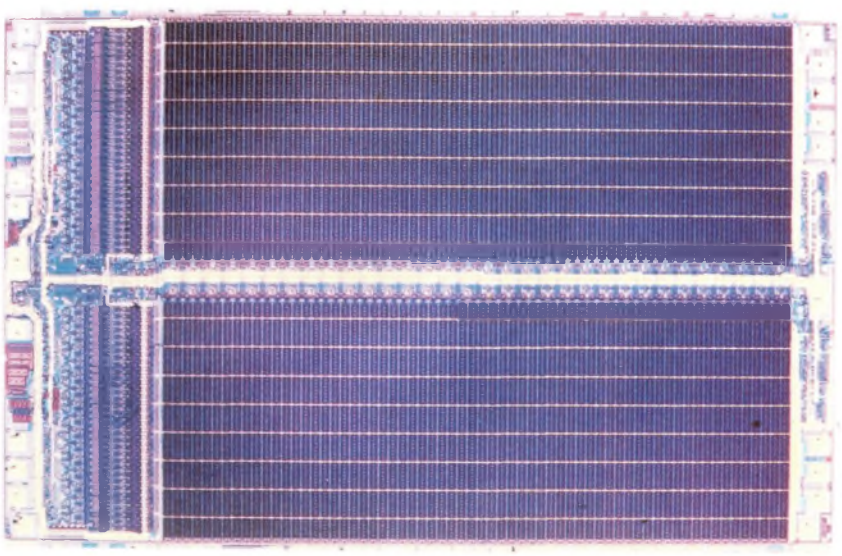
Das organisierte Chaos

Wir brauchen uns nicht darum zu kümmern, wo eine einzelne Information im Gedächtnis abgelegt wird, ebensowenig müssen wir uns darum sorgen, wie diese Information wieder aus dem Gedächtnis hervorgeholt wird. Die genaue Arbeitsweise des menschlichen Gehirns liegt noch im Dunklen, aber wir können uns erinnern, und wenn die Informationen nicht mehr benötigt werden, dann verschwinden sie irgendwie und irgendwo im Gedächtnis.

Nicht so beim Computer. Für ihn ist es 'lebenswichtig' zu wissen, wo die Informationen abgespeichert sind, muß er doch jedes Byte an Information wiederfinden. Dabei spielt es keine Rolle, ob es sich um Programme oder um Daten handelt. Der Computer muß sich also Notizen darüber machen, wo er die einzelnen Daten gespeichert hat.

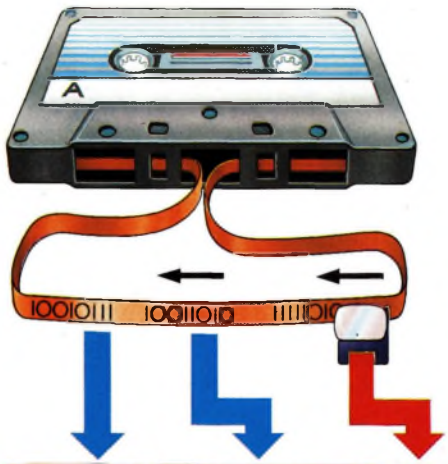
Man kann sich das Computergedächtnis als gigantisches Schubladensystem vorstellen, und jede Schublade – die Speicherzelle – ist mit einer Nummer, ihrer Adresse, versehen. Jede

„Random Access Memory" ist nur eine Variante der vielen elektronischen Gedächtnisse, zu der auch das „Read Only Memory", ROM, gehört. Cassetten oder Disketten sind ein anderes Beispiel für elektromagnetische Gedächtnisse. Das RAM besteht hauptsächlich aus Silizium, welches in einem photochemischen Prozeß zu kleinen Transistoren verarbeitet wird. Für eine Gedächtniszelle, gleichbedeutend mit einem Byte, wird mindestens ein Transistor benötigt. Um in eine der 15797 Gedächtniszellen etwas hineinzuschreiben, braucht unser Computer ca. 200 Nanosekunden, eine fünfmillionstel Sekunde.



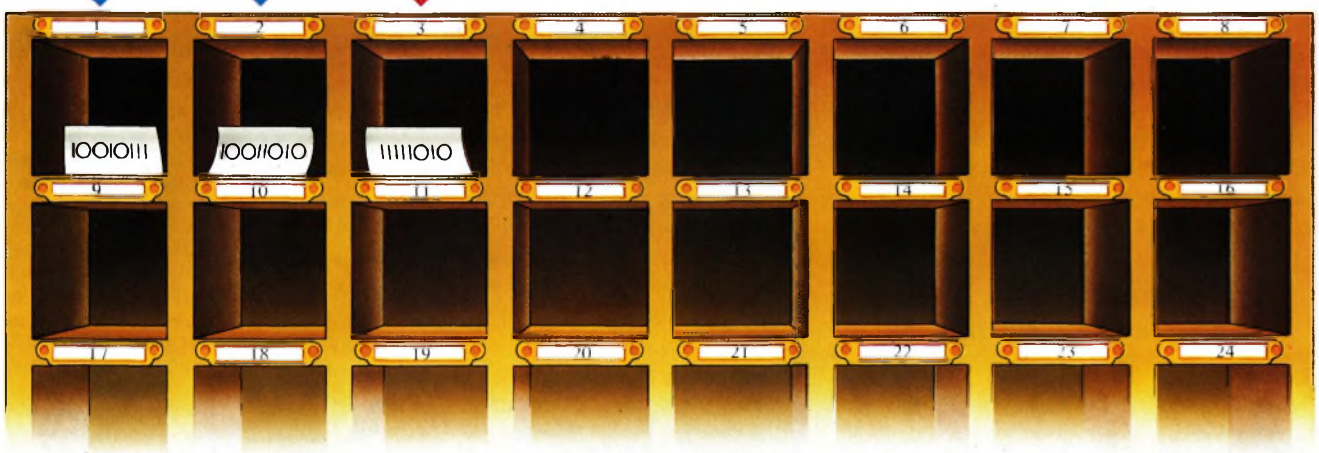
Mostek Ltd.

Das Gedächtnis . . .



Programme, die auf Cassetten gespeichert sind, werden sequentiell aufgenommen. Dies bedeutet, jedes Bit eines Byte wird nacheinander aufgenommen. Wird das Programm wieder in den Computer geladen, liest der Computer jedes Bit, stellt je 8 Bits zu einem Byte zusammen und legt das fertige Byte in eine Gedächtniszelle ab. Das erste Byte kommt in die erste freie Gedächtniszelle, das zweite Byte in die zweite, usw. Soll das so eingelesene Programm ausgeführt werden, muß der Computer nur die Adresse der ersten Gedächtniszelle kennen, die er vollgepackt hat. Der Computer transportiert dann jedes Byte aus den Gedächtniszellen zur CPU, die dann genau das macht, was ihr die Bytes vorschreiben.

Ein Teil des Computergedächtnisses wird vom Betriebssystem belegt. Es ist verantwortlich für grundlegende Arbeiten der CPU, kontrolliert, ob Tasten gedrückt wurden, malt Zeichen auf dem Bildschirm usw. Diese eingebaute Software enthält auch die Programmiersprache BASIC. Und die benötigt Speicherplatz. Nehmen wir an, daß Ihr Computer 64 KByte RAM besitzt und daß das BASIC mit Betriebssystem 16 KByte benötigt, dann haben Sie nur noch 48 KByte zur freien Verfügung. Wenn Sie ein Programm von Cassette in den Computer laden, dann befindet sich das erste Byte des Programms nicht in der ersten, sondern in der ersten freien Gedächtniszelle. Und nur Ihr Betriebssystem weiß, wo diese sich befindet. Wenn Sie das Programm starten, dann sagt das Programm folgendes zur CPU: „Gehe zur Gedächtniszelle X, hole den Inhalt heraus, tue, was der Inhalt sagt, und gehe dann zur nächsten Zelle."



Tony Lodge

Schublade enthält genau ein Byte, nicht mehr und auch nicht weniger. Der Computer findet seine gesuchten Bytes nicht dadurch, daß er alle Schubladen durchsucht, sondern allein anhand der Schubladenadressen. Da der Computer keine Intelligenz besitzt, kann er auch sein Gedächtnis nicht selbst organisieren. Nicht der Computer füllt die Schubladen, sondern der Benutzer stopft die Informationen in der richtigen Reihenfolge und zur richtigen Zeit in die Schubladen hinein. Aber wie geschieht das?

Das Betriebssystem übersetzt

Wenn Sie Ihren Heimcomputer einschalten, dann erscheint normalerweise eine Nachricht auf dem Bildschirm, die Ihnen sagt, daß der Computer nun arbeitet. Diese Nachricht, und die Möglichkeit zu programmieren, ist im Langzeitgedächtnis, dem ROM, des Computers abgespeichert. Dieses ROM enthält aber auch das Betriebssystem. Das Betriebssystem kümmert sich darum, was mit den Eingaben geschieht, es übersetzt BASIC-Kommandos in eine für den Computer verständliche Form oder es malt Buchstaben auf den Bildschirm. Der Computer gibt beispielsweise die Meldung: „15797 bytes free". Diese Information sagt, wieviele Speicherzellen zur Verfügung stehen. Mit jeder Taste, die gedrückt wird, werden die Speicherzellen gefüllt, die Informationen in einer bestimmten Reihenfolge abgelegt. Wird zum Beispiel der Buchstabe K gedrückt, dann wird die binäre Form des Ks in die erste der freien 15797 Speicherzellen abgelegt. Bei einem anschließenden E verschwindet die binäre Form des Es in einer weiteren freien Zelle und ein gedrücktes Y wird entsprechend in eine dritte leere Zelle gespeichert.

In den verschiedenen Speicherzellen ist jetzt das Wort KEY abgelegt. Gleichzeitig ist es auf dem Bildschirm erschienen. Der Computer besitzt ein eingebautes Zählwerk, dessen Zeiger immer genau auf die Zelle zeigt, die gerade gefüllt wird. Allein das Betriebssystem weiß, wo die erste freie Zelle im Computergedächtnis liegt, stellt den Zeiger genau dorthin und setzt das Zählwerk auf Eins. Wird die Speicherzelle Eins gefüllt, wandert der Zeiger zur nächsten freien Zelle und das Zählwerk erhöht sich um Eins auf Zwei. Dieser Vorgang wiederholt sich so lange, bis alle Speicherzellen gefüllt sind oder bis Ihr Programm fertiggestellt ist.

Computerlogik

Microrechner folgen streng den Gesetzen der Logik. Dabei bedient sich der Prozessor sogenannter Gatter – Schaltungen, mit denen die logischen Funktionen AND, OR und NOT dargestellt werden.

Die CPU (Central Processing Unit Zentraleinheit) wird oft als das Herz des Computers bezeichnet. In der Zentraleinheit finden alle Rechenvorgänge und logischen Abläufe statt. Wie aber führt der Computer diese Entscheidungen und Berechnungen durch?

Um dies verstehen zu können, müssen die Grundlagen der binären Arithmetik bekannt sein, und man muß Einblick in die Funktionsweise von logischen Gattern haben. In Computern sind diese Gatter als einfache elektrische Schaltungen installiert, die logische Vergleiche anstellen und Entscheidungen fällen können. Das hört sich komplizierter an, als es in Wirklichkeit ist, denn der Mechanismus dieser Schaltungen läßt sich leicht an Beispielen aus dem täglichen Leben darstellen.

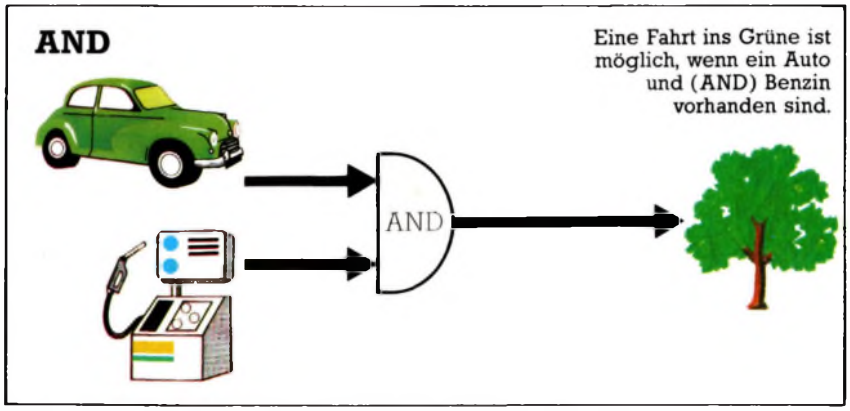
Es gibt drei grundlegende Arten von logischen Gattern: das AND (und)-Gatter, das OR (oder)-Gatter und das NOT (invertierende oder umkehrende)-Gatter.

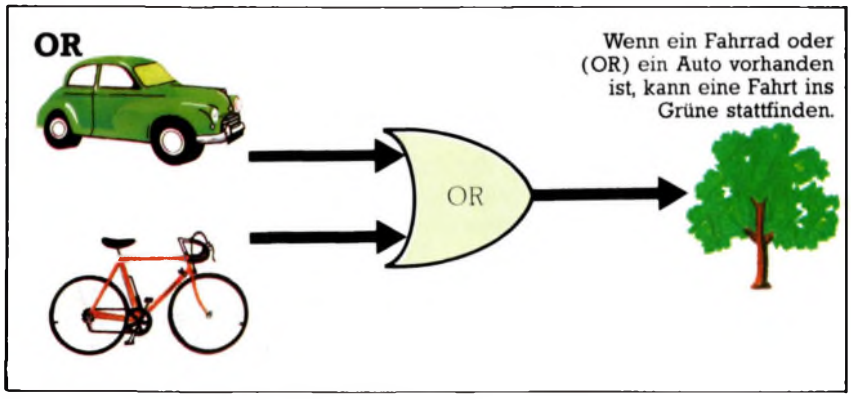
Logische Verbindungen

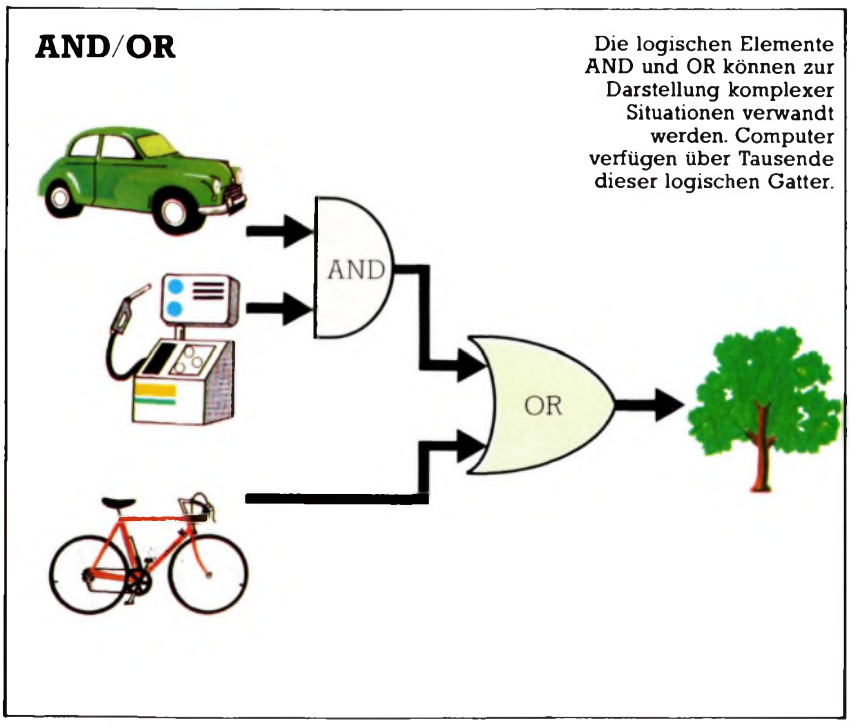
Das AND-Gatter ist eine elektrische Schaltung, die die Information „wahr" ausgibt, wenn alle in diese Schaltung eingeflossenen Informationen „wahr" sind. Ein praktisches Beispiel: Angenommen, Sie möchten ins Grüne fahren. Wenn Sie ein Auto und (AND) Benzin haben, können Sie losfahren. Wenn Sie Benzin haben, aber keinen Wagen, dann können Sie nicht fahren. Ebenso können Sie nicht fahren, wenn Sie einen Wagen, aber kein Benzin haben.

In dieser AND-Schaltung fließen zwei Informationen in die Schaltung ein, die beide „wahr" sein müssen. Um die Fahrt (das „Ergebnis") durchführen zu können, muß es „wahr" sein, daß Sie einen Wagen haben und (AND) es muß „wahr" sein, daß Sie Benzin haben. Dann erst wird das Ergebnis „wahr" – d. h. es ist „wahr", daß Sie eine Fahrt ins Grüne unternehmen können. Wie dieses logische Schema als logische Gleichung dargestellt und in einer „Wahrheitstabelle" aufgelistet werden kann, wird später geklärt.

Stellen Sie sich eine andere Situation vor. Jemand möchte ins Grüne fahren. Die Fahrt ist möglich, wenn er entweder ein Fahrrad oder (OR) ein Auto besitzt. Der Tank des Wagens ist diesmal gefüllt. Wenn er einen Wagen besitzt, dann kann er fahren. Wenn er ein Fahrrad be-







ANDY LESLIE/MARK WATKINSON

sitzt, kann er auch fahren. Nur wenn beide der einfließenden Informationen nicht „wahr" sind, kann die Fahrt nicht stattfinden. – In der Computersprache bedeutet dies: Das Ergebnis ist „falsch", es ist also nicht „wahr", daß er ins Grüne fahren kann.

Es gibt noch ein weiteres grundlegendes logisches Gatter: das NOT-Gatter. Dieses Gatter gibt als Ergebnis das Gegenteil der einfließenden Information aus. Wenn „wahr" die einfließende Information ist, dann ist „falsch" das Ergebnis. Wenn „falsch" die einfließende Information ist, dann ist „wahr" das Ergebnis. In unserem Beispiel einer Fahrt ins Grüne ist es in beiden Situationen „falsch", daß die Fahrt – ob mit dem Auto oder dem Fahrrad – stattfinden kann, wenn ein Reifen geplatzt ist. Ist die einfließende Information (ein geplatzter Reifen) „wahr", dann ist das Ergebnis (eine Fahrt ins Grüne) „falsch".

Diese grundlegenden logischen Elemente können auch miteinander kombiniert werden. Die Erläuterung wieder an unserem Beispiel einer Fahrt ins Grüne. Mit Kombinationen von AND, OR und NOT können alle Entscheidungen, die auf herkömmlicher Logik beruhen, von elektrischen Schaltungen gefällt werden.

Wahrheitstabellen

In den Illustrationen werden die gleichen Symbole verwandt, die auch in der Darstellung von Computerschaltkreisen gefunden werden. Wie einfach logische Entscheidungen mit elektrischen Schaltungen verwirklicht werden können, ist aus der „Wahrheitstabelle" für die Darstellung der AND-Entscheidungen ersichtlich. Mit dem Buchstaben a wird die einfließende Information dargestellt, daß ein Auto vorhanden ist, der Buchstabe b stellt die einfließende Information dar, daß Benzin vorhanden ist, und das Ergebnis – eine Reise ins Grüne – wird mit einem r bezeichnet. Weiterhin werden die Buchstaben W für „wahr" und F für „falsch" benutzt. Die Wahrheitstabelle des logischen Operators AND zeigt alle nur möglichen Kombinationen der einfließenden Informationen und deren Auswirkung auf das Ergebnis:

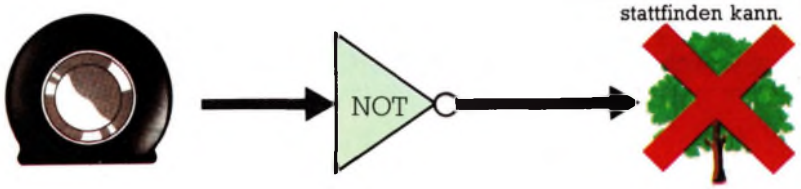
AUTO	F	W	F	W	(a)
BENZIN	F	F	W	W	(b)
REISE	F	F	F	W	(r)

In Computern werden für „wahr" und „falsch" die entsprechenden Binärzahlen 1 und 0 verwandt. Der Computer versteht eine positive Spannung als 1 und eine Nullspannung als 0. Eine AND-Schaltung kann ohne Schwierigkeiten mit Hilfe von Transistoren konstruiert werden. Wenn beide Eingaben positive Spannungsimpulse sind, dann ist das Ergebnis ebenfalls ein positiver Spannungsimpuls. Ist eine oder sind beide Eingaben eine Nullspannung, ist das Ergebnis (output) ebenfalls eine Nullspannung.

Eine elektronische OR-Schaltung erzeugt als Ergebnis (output) eine positive Spannung, wenn einer oder beide Eingangsimpulse positiv sind. Sind beide Eingangsimpulse eine Nullspannung, dann ist das Ergebnis (output) ebenfalls eine Nullspannung. In einer NOT-Schaltung wird der einfließende Impuls in sein Gegenteil umgewandelt: Wenn der einfließende Impuls positiv ist, ist das Ergebnis (output) eine Nullspannung; ist der einfließende Impuls eine Nullspannung, dann ist das Ergebnis ein positiver Spannungsimpuls.

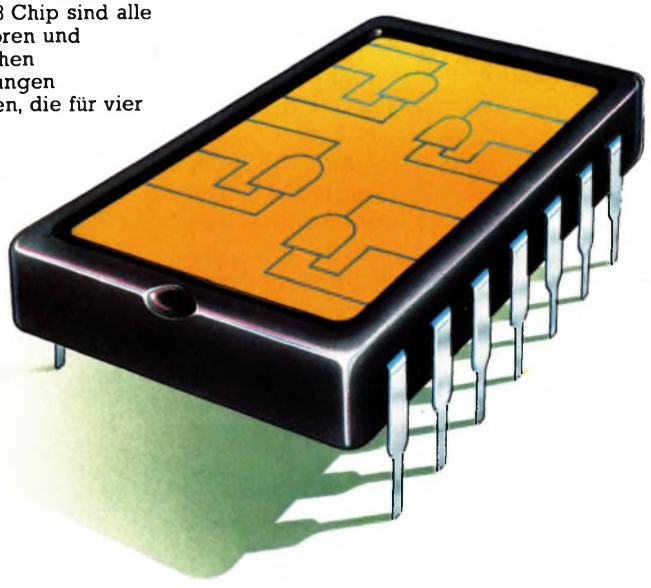
NOT

Ein NOT (invertierendes oder umkehrendes)-Gatter kehrt den Eingangsimpuls in sein Gegenteil um. Wenn es „wahr" ist, daß ein Reifen platt ist, dann ist es „nicht wahr", daß eine Fahrt ins Grüne stattfinden kann.



Der 7408 Chip

Große Siliziumchips (elektronische Bauelemente) enthalten oft Tausende von Gattern, die die logischen Funktionen von AND, OR und NOT ausführen. In dem 7408 Chip sind alle Transistoren und elektrischen Verbindungen vorhanden, die für vier AND-Gatter nötig sind. In der Illustration sind die Gatter als logische Symbole und nicht mit den eigentlichen elektronischen Schaltkreisen dargestellt.



Tony Lodge

Gatter und Addierer

Die Binärzahlen 1 und 0 lassen sich auf einfache Weise mit den logischen Funktionen AND, OR und NOT addieren.

Mit relativ einfachen Transistorschaltungen können die logischen Grundfunktionen AND, OR und NOT realisiert werden. Aus eben diesen Schaltungen ist das Rechenwerk eines Computers aufgebaut. Hier entspricht null Volt dem logischen Wert „falsch" und eine positive Spannung dem Wert „wahr". Man benutzt auch die Symbole Null (0) und Eins (1).

Sind die logischen Grundfunktionen zu Rechengliedern zusammengeschaltet, so bekommen die Nullen und Einsen eine zusätzliche Bedeutung: Sie stehen dann auch gleichzeitig für die Dualzahlen, welche zusammengerechnet werden sollen.

Wenn im Rechenwerk des Computers zwei Dualzahlen addiert werden, gibt es nur vier Möglichkeiten: 0+0, 0+1, 1+0 und 1+1. Wie bereits erläutert, ist 0 plus 0 gleich 0 und 0 plus 1 (bzw. 1 plus 0) gleich 1. Soweit besteht also kein Unterschied zum gewohnten Rechnen mit Dezimalzahlen. Bei der Addition von 1 plus 1 ist zu beachten, daß bei einer Dualzahl eine Stelle höchstens mit 1 besetzt werden kann. Kommt noch eine 1 dazu, steht an dieser Stelle wieder eine 0. Eine 1 wird auf die nächsthöhere Stelle übertragen. Die Tabelle für die Addition von Dualzahlen sieht so aus:

X	Y	Z
0	+ 0 =	0
0	+ 1 =	1
1	+ 0 =	1
1	+ 1 =	10

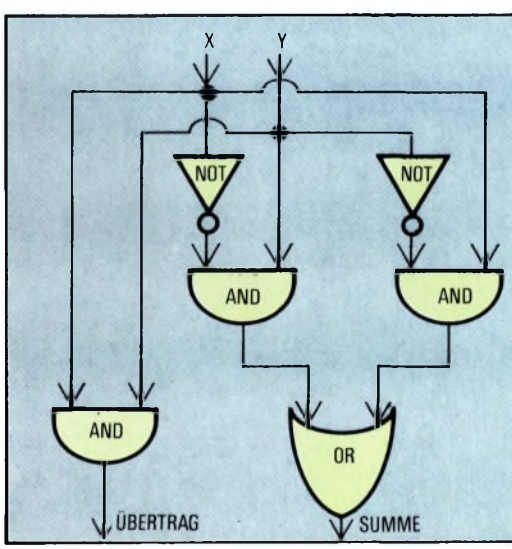
Ein OR-Gatter würde am Ausgang „falsch" (0) zeigen, wenn an beiden Eingängen „falsch" (0) läge. Das Ergebnis der OR-Schaltung wäre „wahr" (1), wenn an einem der Eingänge „wahr" (1) läge. Das würde der Berechnung von 0 + 1 bzw. 1 + 0 entsprechen.

Soweit scheint das einfache OR-Gatter in der Lage zu sein, Dualzahlen zu addieren. Aber für den Fall, daß beide Eingänge „wahr" wären, würde der Ausgang des OR-Gatters auch „wahr" zeigen. Das jedoch entspricht nicht dem erwarteten Ergebnis der binären Addition 1 + 1. 1 + 1 sollte ja 0 mit Übertrag 1 ergeben. Das OR-Gatter erfüllt also nur in drei von vier möglichen Kombinationen die Regeln für die Addition binärer Zahlen, und das ist zu wenig.

Gesucht ist eine Schaltung, die als Ergebnis 0 liefert, wenn beide Eingänge 0 sind, die 1 zeigt, wenn nur ein Eingang 1 ist und die wieder 0 am Ausgang hat, wenn beide Eingänge 1 sind. So verlangt es die Wahrheitswerttabelle für die Addition von Dualzahlen.

Die Erfindung elektronischer Rechenmaschinen ist noch gar nicht so lange her. Bis dahin fanden ausschließlich mechanische Rechenmaschinen (oder Registrierkassen) in Geschäften und Büros Verwendung. Gut 300 Jahre lang ist an der Technik dieser mechanischen Rechenmaschinen praktisch nichts verändert worden. Ihr Innenleben besteht im wesentlichen aus Zahnrädern und ineinandergreifenden Stangen. Die kommerzielle Verwendbarkeit solcher Rechenmaschinen wurde bald erkannt. Pascal erfand die erste Rechenmaschine und verwandte sie im Kontor seines Vaters. Leibnitz entwickelte Maschinen, mit denen es sich auch multiplizieren und dividieren ließ. Von nun an waren die Rechenmaschinen aus dem Geschäftsleben nicht mehr wegzudenken.

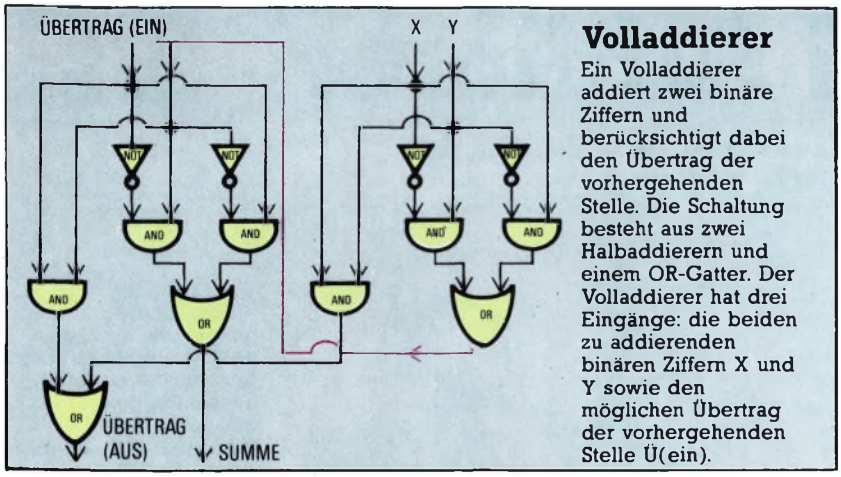
Es gibt eine dementsprechende Schaltung, und sie ist gar nicht so kompliziert. Zwei AND-Gatter, die jeweils mit den beiden Eingängen X und Y verbunden sind, liefern die gewünschten Ergebnisse, wenn der X-Eingang bei dem einen und der Y-Eingang bei dem anderen der beiden AND-Gatter mittels eines NOT-Gatters invertiert wird (siehe Abbildung). Eine 0 an den beiden Eingängen X und Y ergibt „falsch" an den Ausgängen der beiden AND-Gatter. Genauso verhält sich diese Schaltung aber auch, wenn sich an beiden Eingängen eine 1 befindet. Die Ausgänge der beiden AND-Gat-



Halbaddierer

Ein Halbaddierer besteht aus AND-, OR- und NOT-Gattern. Das NOT-Gatter invertiert 1 zu 0 und umgekehrt. Damit ein AND-Gatter 1 am Ausgang zeigt, müssen an beiden Eingängen 1 liegen. Dagegen ist der Ausgang eines OR-Gatters 1, wenn an nur einem der beiden Eingänge oder an beiden zugleich 1 liegt. Der Ausgang eines OR-Gatters ist 0, wenn beide Eingänge 0 sind.

Mark Watkinson



Mark Watkinson

Volladdierer

Ein Volladdierer addiert zwei binäre Ziffern und berücksichtigt dabei den Übertrag der vorhergehenden Stelle. Die Schaltung besteht aus zwei Halbaddierern und einem OR-Gatter. Der Volladdierer hat drei Eingänge: die beiden zu addierenden binären Ziffern X und Y sowie den möglichen Übertrag der vorhergehenden Stelle Ü(ein).

ter zeigen in diesem Fall 0. Wenn einer der Eingänge X und Y „falsch" und der andere „wahr" ist, liegt an einem der beiden AND-Gatter zweimal „wahr". Der Ausgang dieses AND-Gatters ist somit auch „wahr". Die Ausgänge der beiden AND-Gatter sind nun mit den Eingängen eines OR-Gatters verbunden. Dieses OR-Gatter zeigt nur dann „wahr", wenn genau einer der beiden Eingänge X und Y „wahr" ist.

Wahrheitswerttabelle für Halbaddierer

Jetzt fehlt nur noch der Übertrag. Bei der Addition von 1 plus 1 muß außer der 0 am Summenausgang eine 1 an der Übertragsstelle erscheinen. Dazu ist ein weiteres AND-Gatter mit den Eingängen X und Y verbunden. Wenn X und Y „wahr" sind, ist auch der Ausgang dieses AND-Gatters „wahr". Die Wahrheitswerttabelle für den abgebildeten Halbaddierer sieht folgendermaßen aus:

X (Eingang 1)	Y (Eingang 2)	Ü (Übertrag)	S (Summe)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

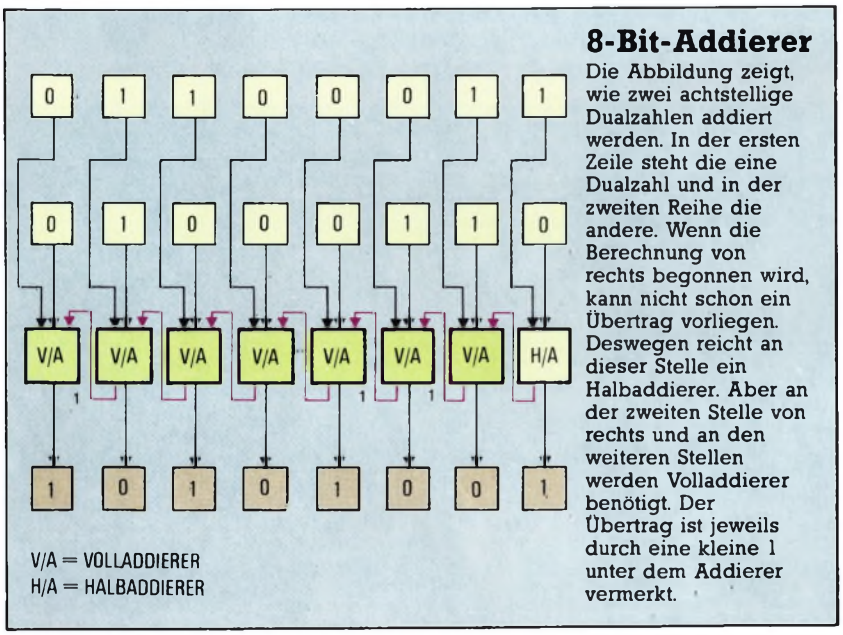
Diese Schaltung wird als Halbaddierer bezeichnet, da sie zwar zwei Ziffern addieren kann, aber den Übertrag einer vorhergehenden Stelle nicht berücksichtigt. Normalerweise werden immer gleich ganze Bytes addiert. Und ein Byte besteht aus acht Bits. Die Addierschaltung für die ganz rechts stehende erste Stelle der Dualzahl muß tatsächlich nur ein Halbaddierer sein. Die restlichen Addierer dagegen müssen neben den beiden zu addierenden binären Ziffern noch den möglichen Übertrag aus der rechts danebenstehenden Spalte berücksichtigen können. Man betrachte das folgende Additionsbeispiel:

```
 011
+111
1010
```

Bei der Einerstelle rechnen wir 1 plus 1 gleich 0 Übertrag 1, schreiben die 0 und merken uns die 1. Bei der Zweierstelle rechnen wir wieder 1 plus 1 gleich 0 Übertrag 1. Jetzt müssen wir aber noch den Übertrag aus der Einerstelle hinzuziehen. Es resultiert also 1 Übertrag 1. Wir schreiben die 1 an die Zweierstelle und merken uns den Übertrag für die Viererstelle. Hier rechnen wir 0 plus 1 gleich 1. Zu dieser 1 addieren wir den Übertrag aus der Zweierstelle und erhalten 0 Übertrag 1. Wir schreiben die 0 an die Viererstelle und die 1 des Übertrags an die Achterstelle. Wird der Übertrag der vorhergehenden Stelle berücksichtigt, ergibt sich ein neues Bild auf der Wahrheitstabelle. Die Tabelle für einen Volladdierer sieht dann folgendermaßen aus:

X	Y	Ü(ein)	Ü(aus)	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Ein Volladdierer besteht aus zwei Halbaddierern und einem OR-Gatter. Der ausgangsseitige Übertrag eines Volladdierers ist direkt mit dem eingangsseitigen Übertrag des links nebenstehenden Volladdierers verbunden. Auf diese Weise können beliebig viele Volladdierer verkettet werden. Die Berechnung wird in zwei Schritten ausgeführt. Zuerst werden die beiden Ziffern addiert und dann wird der Übertrag hinzugerechnet.



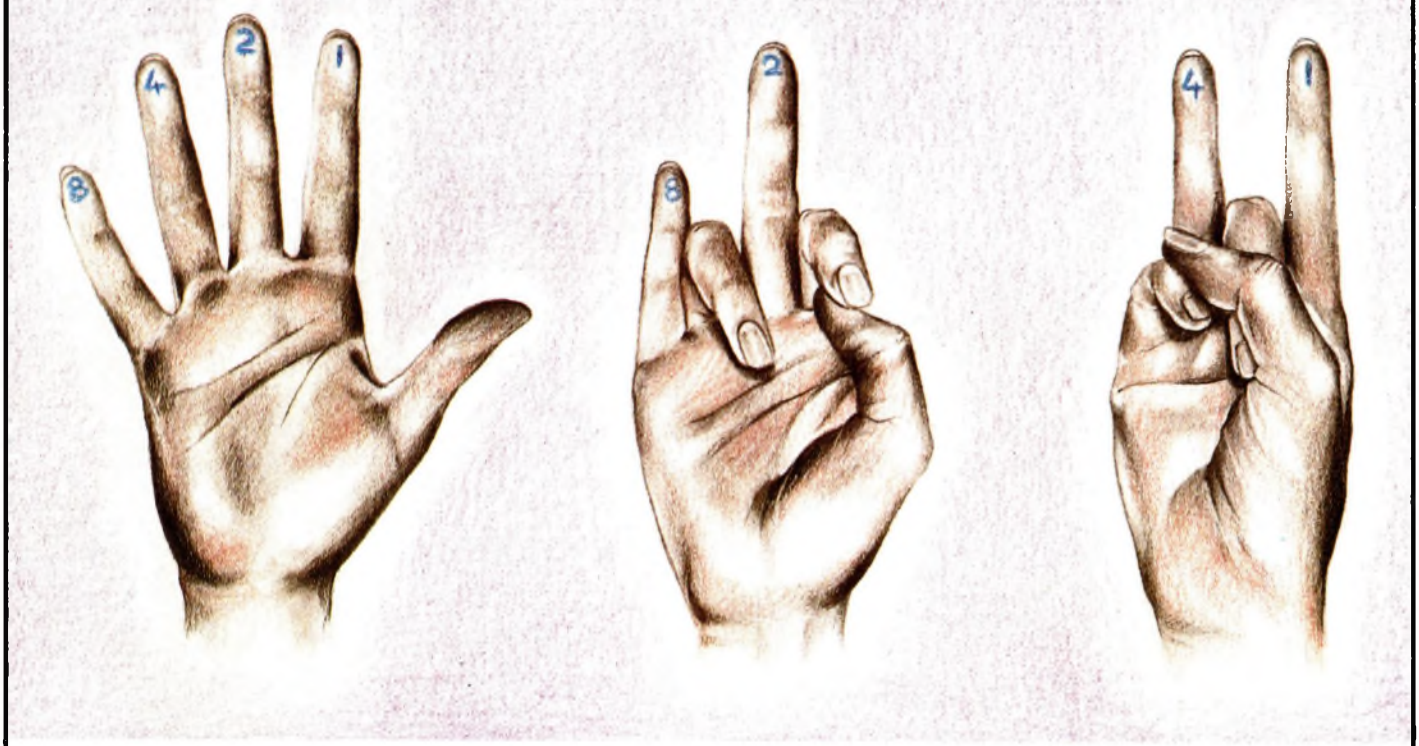
8-Bit-Addierer

Die Abbildung zeigt, wie zwei achtstellige Dualzahlen addiert werden. In der ersten Zeile steht die eine Dualzahl und in der zweiten Reihe die andere. Wenn die Berechnung von rechts begonnen wird, kann nicht schon ein Übertrag vorliegen. Deswegen reicht an dieser Stelle ein Halbaddierer. Aber an der zweiten Stelle von rechts und an den weiteren Stellen werden Volladdierer benötigt. Der Übertrag ist jeweils durch eine kleine 1 unter dem Addierer vermerkt.

Wenn 1+1=10 . . .

Mit den Binärziffern 0 und 1 kann der Computer jede Dezimalzahl darstellen. So entspricht zum Beispiel 111111010 der Zahl 506.

Die Umwandlung ins Binäre



Ein einfacher Weg, vierstellige binäre Zahlen in Dezimalzahlen zu verwandeln, ist dieser: Auf die Finger der rechten Hand schreibt man (z. B. per Aufkleber) die Stellenwerte der Binärzahlen (8, 4, 2, 1). Finger, die binär Eins sind, streckt man aus, Finger, die binär Null sind, bleiben gekrümmt. Wenn wir dies für binär 1010 ausführen, erhalten wir 8 und 2, was zusammen dezimal 10 ergibt. Für binär 0101 erhalten wir auf gleichem Wege dezimal 5.

Versuchen Sie es einmal selbst. Welche Dezimalzahlen erhalten Sie für binär 1110 und 0110? Wir können diese Methode bis auf achtstellige Binärzahlen ausdehnen, wenn wir die Finger der linken Hand mitverwenden. Sie repräsentieren dann die Stellenwerte 16, 32, 64 und 128, wobei wir dem kleinen Finger 16 und dem Zeigefinger 128 zuordnen. Wie stellen Sie mit Hilfe aller Finger die Zahl 180 dar?

Elaine Keenan

Für viele Menschen sind Dezimalzahlen etwas so Selbstverständliches, daß sie sich ein anderes Zahlensystem gar nicht vorstellen können. Dabei erdachten sich die Römer ein System, in dem Zahlen durch Buchstaben ausgedrückt werden. So stellte X die Zahl 10, L die Zahl 50, C die Zahl 100, D die Zahl 500 dar, um nur einige zu nennen. Für die Aufzeichnung einfacher Zahlen funktionierte dieses System ganz gut, zum Rechnen aber taugte es nicht viel. Selbst einfache Additionen fallen mit römischen Zahlen schwer, weil eine Zuordnung von Stelle und Wert fehlt.

Sieht man sich aber die beiden Dezimal-Zahlen 506 und 56 an, so fällt als erster Unterschied die Null in 506 auf. Sie sagt uns, daß 506, neben fünf Hundertern und sechs Einern, keine Zehner hat. In unserem Zahlensystem hat jede Ziffer ihren Stellenwert. In der ersten Stelle von rechts stehen die Einer, in der nächsten nach links stehen die Zehner, dann die Hunderter, die Tausender und so weiter. Die Ziffer gibt also nur an, wieviele „Werte" in einer Stelle stehen.

Was aber hat dies alles mit Computern und dem binären Zahlensystem zu tun? Nun, Computer sind elektronische Maschinen, die mit Zahlen gut umgehen können, sobald sie durch Spannungspegel repräsentiert werden. So kann 5 Volt binär Eins und 0 Volt binär Null darstellen. Sie wissen, daß Einsen und Nullen im Binärsystem völlig ausreichen, um jede beliebige Zahl, egal wie groß, darzustellen.

Allerdings, so kompakt wie unser Dezimal-

oder Zehnersystem ist das binäre System nicht. So ist die Dezimalzahl 506 in binärer Form 111111010 für uns wesentlich unhandlicher. Das liegt daran, daß im Dezimalsystem die Stellenwerte Potenzen von 10 und im Binärsystem Potenzen von 2 sind. Hinzu kommt, daß im Dezimalsystem pro Stellenwert zehn Ziffern, von 0 bis 9, zur Verfügung stehen, im Binärsystem dagegen pro Stellenwert nur zwei, nämlich 0 und 1. Der geringere Ziffernvorrat führt also im Binärsystem zu mehrstelligeren Zahlen. Benötigt der Wert Zehn dezimal zwei Stellen, so sind es im Binärsystem bereits vier: 1010.

Obwohl zwischen beiden Zahlensystemen grundsätzliche Ähnlichkeiten bestehen (beide sind Stellenwert-Systeme) und der kleinste ganzzahlige Wert bei beiden Eins ist, sind die Ergebnisse in Ziffern doch sehr unterschiedlich. Wir wollen dies am Beispiel der Dezimalzahl 506 weiter verdeutlichen:
Im Dezimalsystem:

Hunderter	Zehner	Einer	
5	0	6	
= 5×100	+ 0×10	+ 6×1	= 506

Im Binärsystem:

256er	128er	64er	32er	16er	8er	4er	2er	1er	
1	1	1	1	1	1	0	1	0	
= 1×256	+1×128	+1×64	+1×32	+1×16					
+1×8	+0×4	+1×2	+0×1						= 506 dezimal

Die uns vom Dezimalsystem her vertrauten Rechenregeln gelten genauso im Binärsystem. Wir müssen nur beachten, daß hier die größte Ziffer 1 und nicht, wie gewohnt, 9 ist. Zum Beweis einige Additionen. (Zur Unterscheidung stehen Dezimalzahlen in Klammern):

(3)	11
+(5)	+101
(8)	1000

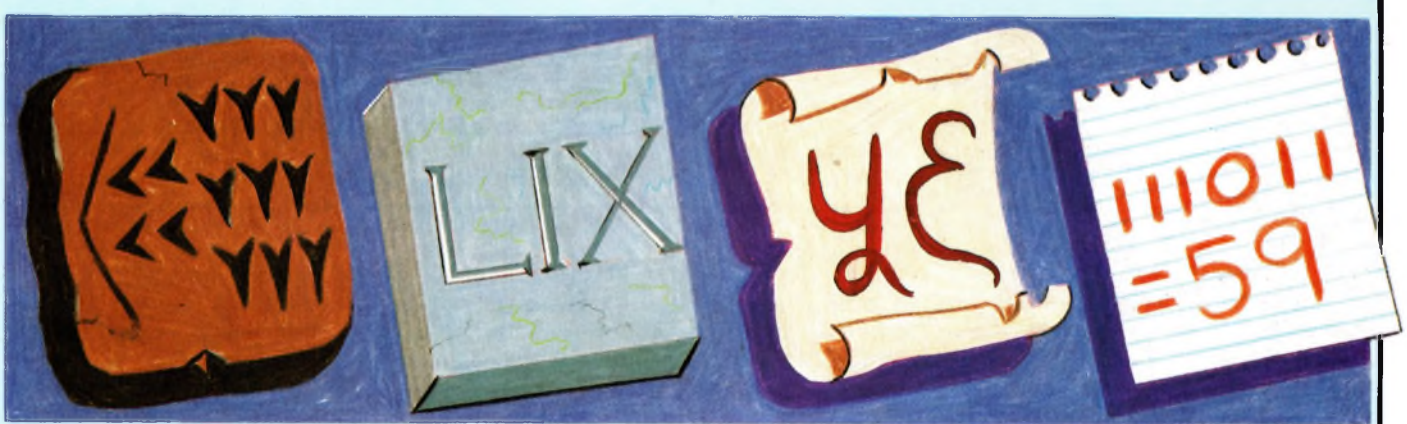
1+1=0 und Übertrag 1
Übertrag 1+0+1 = 0 und Übertrag 1
Übertrag 1+1 = 0 und Übertrag 1
Übertrag 1+0 = 1

Wir haben gerade gesehen, daß binär Eins und Eins gleich Null mit Übertrag Eins ergibt (dies ist vergleichbar zu 9 und 1 gleich 10, „Null hin Übertrag Eins im Sinn"). Im Binärsystem ist Eins die größte Ziffer, vergleichbar zu Neun im Dezimalsystem. Hier nun die nächste Addition und zwei weitere Beispiele zum Ausprobieren:

(4)	100		(7)	111	(3)	11
+(6)	+110		+(2)	+ 10	+(12)	+1100
(10)	1010		(?)	?	(?)	?

Spätestens jetzt haben Sie feststellen können, daß binäre Zahlen viel länger als ihre gleichwertigen dezimalen Partner sind. Versuchen Sie doch einmal, binär 11010110 und 1101101 zu addieren. Vergessen Sie aber nicht, gleiche Stellenwerte untereinander zu schreiben, genau wie beim Rechnen mit Dezimalzahlen.

Die Geschichte der Zahlensysteme



Babylonisch **Römisch** **Hindustanisch** **Binär**

Die alten Babylonier entwickelten vor mehr als 4000 Jahren ein Zahlensystem, das, im Gegensatz zu unserem heutigen Dezimalsystem, die Zahl 60 als Basis verwendete. Die Illustration zeigt die Zahl 59 in babylonischer Keilschrift. Das römische Zahlensystem war im Vergleich zum Babylonischen ein Rückschritt. Buchstaben repräsentierten Zahlen. Weil aber die Stellung der Buchstaben-Zahl keinen Hinweis auf ihren Wert gab, waren selbst einfache Rechenoperationen fast nicht ausführbar. Die Hindus verwendeten neun Zeichen für Zahlen von 1 bis 9 und fügten später ein Zeichen für 0 hinzu. Ihr Beitrag zur Entwicklung der Zahlensysteme war die Einführung des Stellenwertes: Der Wert eines Zeichens (Ziffer) richtete sich nach seiner Stellung. Das System wurde von den Arabern übernommen und breitete sich von dort nach Europa aus. Einer der führenden arabischen Mathematiker war Al-Huwarizmy. Aus der latinisierten Form seines Namens entstand der Begriff Algorithmus und aus der Abkürzung des Titels seines Lehrbuches „Al-Gabr Wal-Muqabala" entstand der Begriff Algebra. Die Vorliebe der Computer für das Binärsystem hat ihren Grund darin, daß sich jede beliebige Zahl mit Einsen und Nullen darstellen läßt.

Andy Leslie

Von Minus zu Plus

Da Computerschaltungen einfach aufgebaut sein müssen, werden Subtraktionen mit Hilfe eines kleinen Tricks nach den Regeln der Addition durchgeführt.

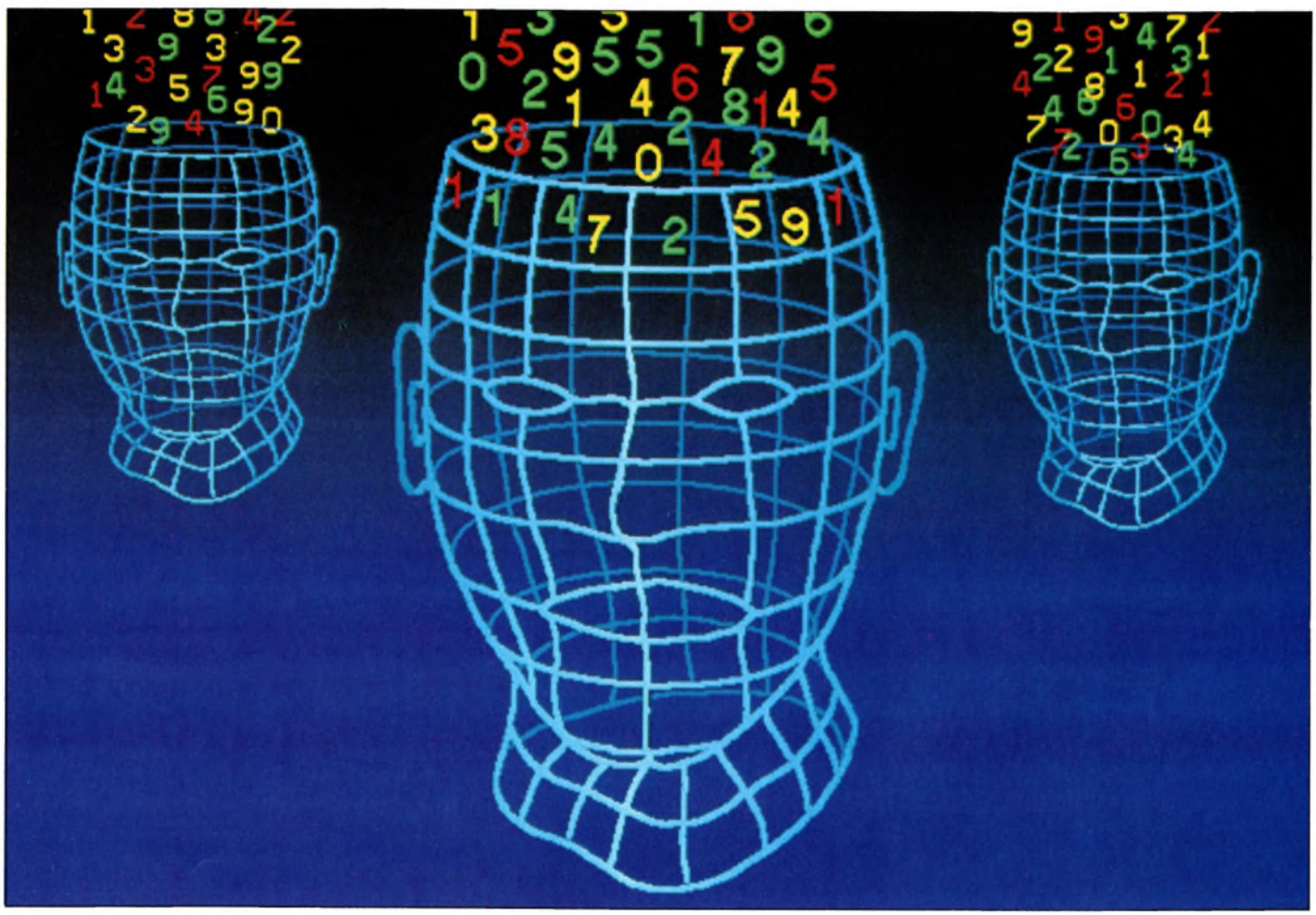
Auf dem Papier können Binärzahlen nach den gleichen Regeln wie Dezimalzahlen subtrahiert werden. Die Konstrukteure von Computern hatten aber schon vor langer Zeit entdeckt, daß elektronische Schaltungen, die Additionen ausführen, auch subtrahieren können, ohne daß dafür eine besondere Schaltung benötigt würde.

Das „Zweierkomplement" ist eine Methode, negative Zahlen in Computern darzustellen. Hierbei scheint ein Subtraktionsvorgang nur Teil einer normalen Addition zu sein, wie beispielsweise bei dieser Aufgabe:

$$16 - 12 = 4$$
oder $16 + (-12) = 4$

Von der Zahl 16 wird die Zahl 12 abgezogen. Der Vorgang der Subtraktion kann aber auch als Addition betrachtet werden: Zu der positiven Zahl 16 wird die negative Zahl −12 hinzugezählt. Beide Lösungswege führen zum gleichen Ergebnis, der einzige Unterschied liegt in der Verwendung arithmetischer Vorzeichen und Klammern. Diese geringfügige Umstellung verwendet ein Computer, um negative Zahlen darzustellen und die Probleme der Subtraktion damit zu vereinfachen.

Zur Erläuterung dieser Methode nehmen Sie einmal an, Sie hätten einen einfachen Rechner, der nur Zahlen mit maximal fünf Stellen verarbeiten kann. (Echte Computer können natürlich Zahlen mit Tausenden von Stellen verarbeiten.) Der 5-Stellen-Computer funktioniert also folgendermaßen – die führende Stelle auf der linken Seite wird anders behandelt als die restlichen vier: Ist die führende Stelle links eine 1, dann stellt sie die negative Zahl −16



dar; ist sie 0, dann ist ihr Wert ebenfalls 0. Die anderen vier Stellen stellen positive Zahlen dar, die nach den normalen binären Regeln funktionieren, die bereits behandelt wurden.

[]	[]	[]	[]	[]
−16 oder 0	8	4	2	1 oder 0

Die binäre Zahl 01000 ist also die Dezimalzahl 8 und binär 10 000 ist dezimal −16. Was aber bedeutet 10100? Die Binärzahl bedeutet dezimal sowohl −16 als auch +4 und ergibt daher −12.

Die höchste positive Zahl, die mit diesem System dargestellt werden kann, ist 01111 oder dezimal 15. Die größte negative Zahl ist 10000 oder dezimal −16. Mit ein wenig Probieren kann man damit jede Zahl zwischen −16 und +15 darstellen.

Binär Dezimal
10000 −16
10001 −15
10010 −14
10011 −13
10100 −12
etc.
11111 −1
00000 0
00001 1
00010 2
etc.
01110 14
01111 15

Wenn die Stellenzahl, die der Computer verarbeiten kann, vergrößert würde, könnte man den verfügbaren Zahlenbereich ausdehnen.

In der Entwicklung der binären Arithmetik der Computer hat man schon früh einen einfachen Trick entdeckt, mit dem das Zweierkomplement (oder: die negative Form einer Zahl) gefunden werden kann.

Zuerst wird jede Stelle in ihr Gegenteil verkehrt. Steht an einer Stelle eine 1, so wandelt sie sich in eine 0, und entsprechend wandelt sich eine 0 in eine 1. In der zweiten Stufe wird zu dieser umgekehrten Zahl eine 1 addiert.

Folgendes Beispiel erläutert diesen Vorgang: Die binäre Entsprechung der Zahl +12 ist 01100 (die führende 0 auf der linken Seite ist eigentlich nicht nötig, da 01100 das gleiche ist wie 1100. Da der Computer aber fünf Stellen hat, muß jede Stelle ausgefüllt werden). Das Zweierkomplement wird dann folgendermaßen gebildet:

01100 (= +12)

Erste Stufe: 10011
Zweite Stufe: 00001 (+1)
10100 (= −12)

Im folgenden Beispiel wird gezeigt, wie der Computer damit eine Subtraktion ausführt – zum Beispiel: 12 − 4.

+12 ist 01100
−4 ist 11100 (Zweierkomplement)
12+ (−4) 101000

Das Ergebnis hat jetzt sechs Stellen. Da der Computer aber nur fünf Stellen verarbeiten kann, bezeichnet man die führende sechste Stelle als „Überlauf" und ignoriert sie. Das Ergebnis ist damit 01000 oder dezimal 8. Ein etwas komplizierteres Beispiel ist: 4 minus 12.

+4 ist 00100
−12 ist 10100
4+ (−12) 11000

Im letzten Beispiel werden zwei negative Zahlen zusammengezählt:
−3 −4 = −3+ (−4) = −7

+3 ist 00011
−3 ist 11101 (Zweierkomplement)
−4 ist 11100
111001

Hier erhält man wieder eine sechsstellige Zahl als Ergebnis. Ohne den Überlauf erhält man die Binärzahl 11001 oder dezimal −7.

Diese Art der Subtraktion verwendet nur die Addition und den Trick des Zweierkomplements (der nur die Umkehr der einzelnen Stellen und wiederum die Addition anwendet) und gibt damit dem Computer die Möglichkeit, Binärzahlen über ein NOT-Gatter auf einfache Weise in ihre Komplemente umzusetzen.

Ein NOT-Gatter ist eine logische Schaltung, die über eine Eingangs- und eine Ausgangsleitung verfügt. Ihre Funktion ist es, eingehende Impulse in ihr Gegenteil zu verkehren. Eine eingehende 1 wird in eine ausgehende 0 gewandelt und eine eingehende 0 als eine 1 wieder ausgegeben. Diese ‚Invertierung' (Umkehrung) verwendet der Computer in der ersten Stufe der Herstellung eines Zweierkomplements.

Morsezeichen sind eines der ersten Beispiele für binäre Codierung im Bereich der Elektronik. 1837 wurde in London zwischen den beiden Bahnhöfen Euston und Camden Town die erste telegrafische Verbindung der Welt mit einem Kabel von drei Kilometern Länge hergestellt. Später im selben Jahr führte Samuel Morse in Amerika das Morsealphabet vor, in dem jeder Buchstabe aus einer Kombination der Signale Punkt und Strich zusammengesetzt wird.

Die Schaltzentrale

In der sogenannten CPU, der Zentraleinheit, werden alle Funktionen des Computers gesteuert. Sämtliche Arbeitsprozesse werden von der CPU übernommen und an die dafür vorgesehenen Chips verteilt.

Alle Funktionen eines Computers werden von der Zentraleinheit (Central Processing Unit—CPU) gesteuert. Ein sehr einfacher Computer (siehe Bild) müßte nur aus der Zentraleinheit, Speicher und einigen E/A-Schaltungen bestehen. Ein- und Ausgabemechanismen (E/A) benötigt der Computer für die Kommunikation mit der „Außenwelt".

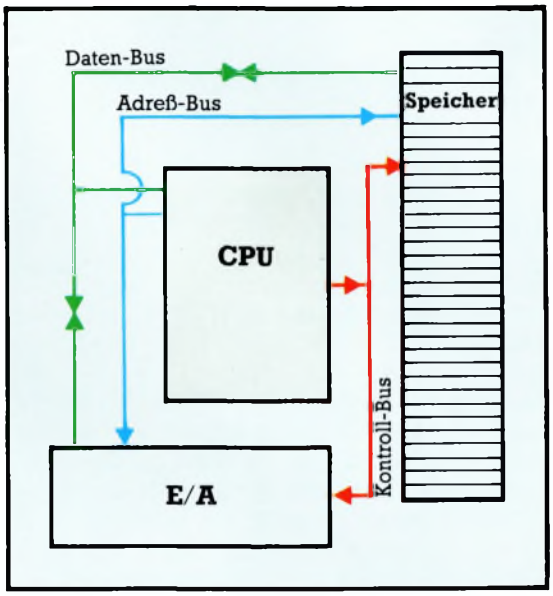
Ein Speicher wird für die Bereitstellung von Instruktionen und Daten für die Zentraleinheit gebraucht. Die zu bearbeitenden Daten können dabei sowohl Zahlen als auch binären Code enthalten, der eine Vielfalt von Zeichen darstellen kann.

Somit enthalten einige der Speicherstellen Befehle, andere dagegen Daten, die die Zentraleinheit bearbeiten soll. Wie kann die Zentraleinheit zwischen diesen beiden verschiedenen Speicherinhalten unterscheiden?

Zentraleinheiten der 8-Bit-Prozessoren (die Steuereinheit fast aller Heimcomputer) werden normalerweise in Form eines einzigen Chips gebaut, bei dem sich an beiden Längsseiten 20 Kontaktstifte befinden. Über jeden dieser Kontaktstifte (abgesehen von den zwei Stiften für die Stromversorgung von 0 und +5 Volt) laufen Impulse, die von Peripheriegeräten wie E/A-Platinen oder von dem Speicher übermittelt oder dorthin gesendet werden.

Typische Zentraleinheiten verfügen über 16 Adreßkontakte, die mit dem Adreß-Bus verbunden sind. Jeder dieser Kontaktstifte übermittelt ein Ausgabesignal, das eine Null oder eine Eins darstellen kann. Die sechzehn Adreßleitungen können also 65536 Kombinationen dieser beiden Zeichen darstellen und damit die gleiche Anzahl von Speicherstellen individuell ansprechen.

Weiterhin gibt es acht Datenkontakte, die mit dem Daten-Bus verbunden sind. Diese Kontakte übermitteln von einer Ein- und Ausgabeeinheit oder von dem Speicher Daten an die Zentraleinheit oder empfangen sie von dort. Weitere Kontakte übermitteln Kontrollsignale, die entweder von der Zentraleinheit erzeugt oder dorthin gesendet werden.



Ein sehr einfacher Computer (siehe Bild) braucht eigentlich nur aus einer Zentraleinheit, Speicher und einer E/A-Schaltung zu bestehen. In dem Speicher sind Befehlsfolgen enthalten, die über die Zentraleinheit bestimmte Abläufe auslösen. Der Speicher kann ebenfalls Daten unterbringen, die von der Zentraleinheit später benötigt werden. Die E/A-Schaltung wird zur Kommunikation mit der Außenwelt benötigt. Steuert ein Computer beispielsweise eine Waschmaschine, dann erhält er seine Eingabesignale über die Knöpfe der Frontplatte und seine Ausgabesignale schalten Motoren und Heizung aus und ein. Die internen Befehlsfolgen der Zentraleinheit sind binär codiert. Die verschiedenen Typen von Zentraleinheiten arbeiten alle mit unterschiedlichen Befehlssätzen.

Universalregister

Innerhalb der Zentraleinheit gibt es einige Spezialspeicher mit einem oder zwei Byte Kapazität, sogenannte Register. Die anderen Speicher innerhalb der Zentraleinheit werden Universalregister genannt und für die Zwischenspeicherung von Informationen verwendet. Weiterhin gibt es noch zwei Funktionsmodule: die ALU und den Kontroll-Block.

Die Abkürzung ALU steht für Arithmetik-Logik-Einheit (englisch: Arithmetic and Logic Unit). Dieser Teil der Zentraleinheit führt die mathematischen und logischen Vorgänge aus, wie Additionen, das Rotieren von Bits innerhalb eines Bytes und die Abläufe bei den logischen Operatoren AND und OR.

Der Kontroll-Block ist eine spezielle Schaltung, die die Impulsfolgen des Systems koordiniert und synchronisiert. Die Funktion wird an dem Beispiel der weitverbreiteten Z80-Zentraleinheit erläutert.

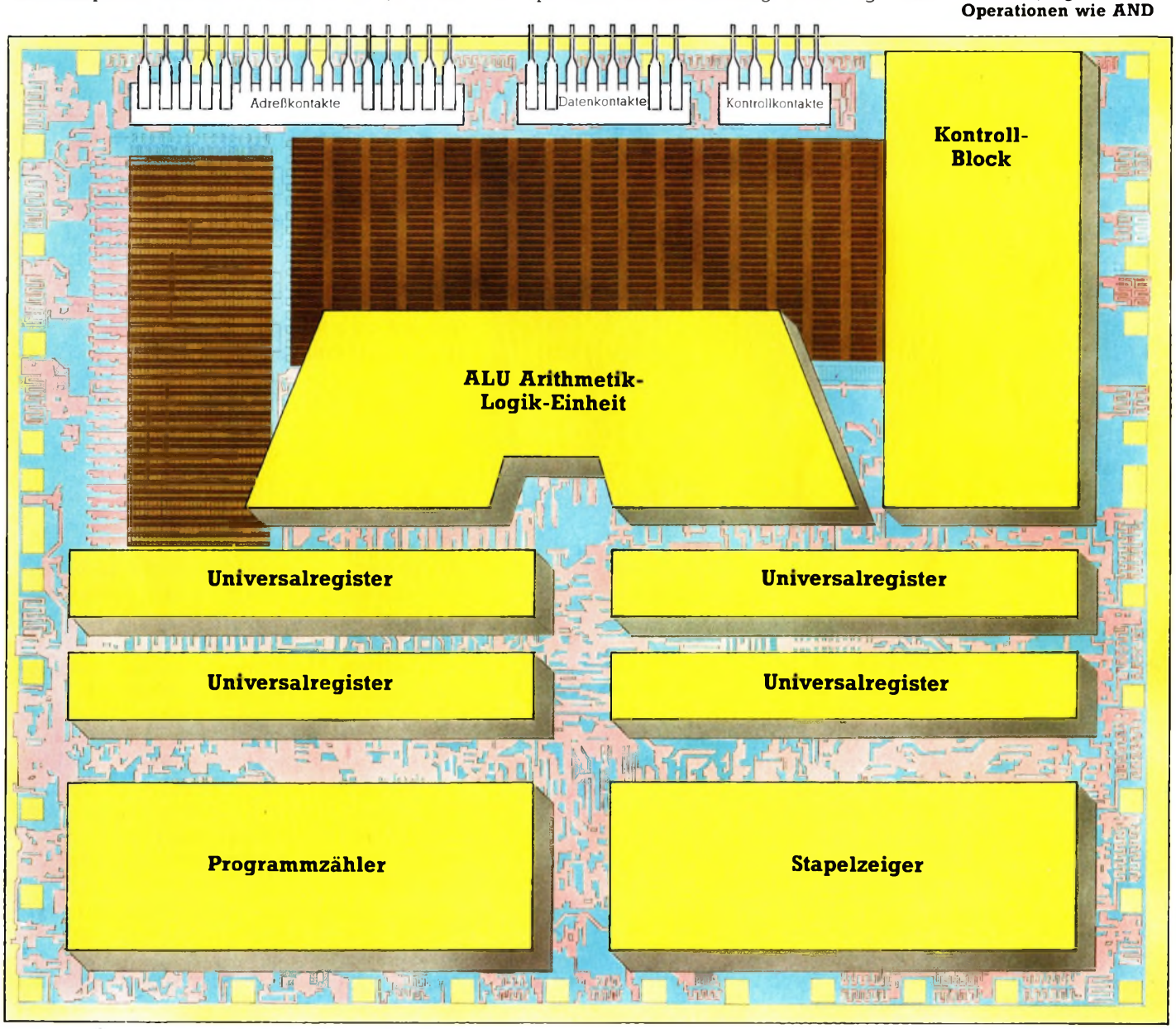
Wird von der Zentraleinheit der Befehl 11000110 aus dem Speicher empfangen, so addiert sie den Inhalt des nächsten Bytes im Speicher auf den Inhalt eines ihrer internen Register. Wenn das Resultat an einer bestimmten Speicherstelle abgelegt werden soll, muß der nächste Befehl an die Zentraleinheit die Zahlenfolge 00110010 sein, gefolgt von zwei Bytes, die die Speicherstelle bezeichnen.

Angenommen, das Berechnungsergebnis ist die Dezimalzahl 37 und die zwei Bytes hinter dem Befehl bezeichnen die Speicherstelle 33126 (dezimal). Der Befehl veranlaßt jetzt den Kontroll-Block, die Adreßkontakte auf die binäre Entsprechung von 33126 (d. h.

1000000101100110) zu setzen. Weiterhin wird dem Speicher über den Kontroll-Bus das Signal gesendet, daß in Kürze Daten ankommen werden, die gespeichert werden sollen. Über den Daten-Bus werden die Daten dann an den Speicher gesandt und auf der Speicherstelle abgelegt, die von dem Adreß-Bus übermittelt wurde. Werden diese Daten zu einem späteren Zeitpunkt von der Zentraleinheit (z. B. für ihre Darstellung auf dem Bildschirm) benötigt, müßte ein anderer Befehl an die Zentraleinheit übermittelt werden. In Kurzform würde der Kontroll-Block diesen Befehl als „von der Speicherstelle 33126 ein Byte holen und in einem der internen Register zwischenspeichern" verstehen.

Die Anzahl der internen Register oder Zwischenspeicher einer Zentraleinheit hängt von ihrem Typ ab. Es gibt 8-Bit- (ein Byte) oder 16-Bit- (zwei Byte) Register. Die Spezialregister haben normalerweise Namen wie Stack Pointer, Program Counter oder Akkumulator. Universalregister werden als X-Register, Y-Register oder C-Register bezeichnet.

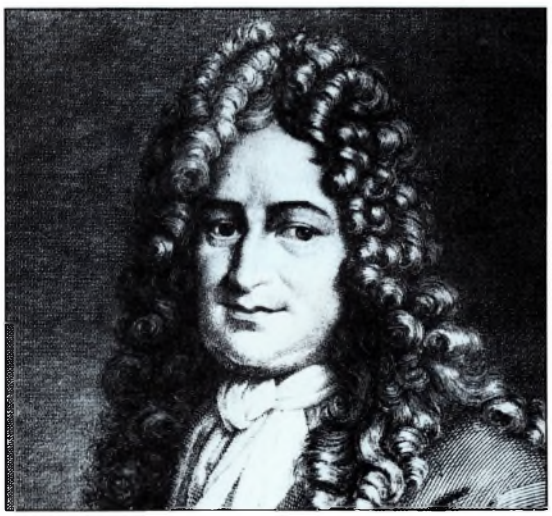
Eines der wichtigsten Spezialregister ist der Program Counter, der Programmzähler. Diese interne Speicherzelle enthält (in Binärzahlen) immer die Adresse des nächsten Befehls, der im Speicher zur Ausführung bereitliegt. Will die Zentraleinheit den nächsten Befehl ausführen, wird der Inhalt des Programzählers über den Adreß-Bus an den Speicher ausgegeben und das an der entsprechenden Stelle gespeicherte Byte über den Daten-Bus der Zentraleinheit übermittelt. Ein wichtiger Teil der 8-Bit-Register ist der Akkumulator. Der Akkumulator ist ein Zwischenspeicher für von der ALU erzeugte Ergebnisse oder für Daten, die vom Speicher oder den E/A-Bausteinen übermittelt wurden oder dorthin gesendet werden sollen.



Die Abbildung zeigt eine Zentraleinheit mit ihren Registern, einer Arithmetik-Logik-Einheit, die aus Hunderten von logischen Gattern besteht (mit denen Rechenoperationen wie Addition, logische Operationen wie AND und die Herstellung des Komplements von Binärzahlen ausgefuhrt werden) und dem Kontroll-Block. Der Kontroll-Block nimmt die (binär) codierten Befehle entgegen, interpretiert sie und steuert dann die Abläufe innerhalb der Zentraleinheit.

Multiplizieren im Binärsystem

Computer multiplizieren auch die größten Zahlen durch einfaches Zusammenzählen von Binärziffern.



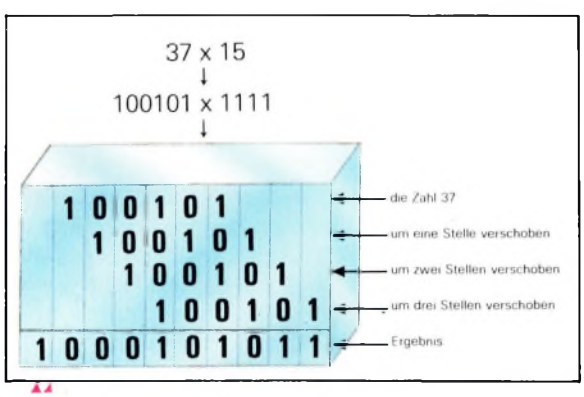
Gottfried Wilhelm Leibnitz (1646–1716) war ein Zeitgenosse von Isaac Newton und befaßte sich mit Philosophie, Mathematik und Technik. Er untersuchte die Möglichkeiten der Anwendung binärer Arithmetik in Vorrichtungen zum Rechnen und konstruierte eine Maschine zum Multiplizieren und Dividieren. In späteren Jahren diskutierte Leibnitz mit Newton über Themen der höheren Mathematik.

Will man 14 mit 12 multiplizieren, wäre der einfachste Weg, die 14 zwölfmal untereinander zu schreiben und dann zu addieren, also 14 + 14 + 14 + ... (zwölfmal). Da die Multiplikation in gewisser Weise ein wiederholtes Addieren ist, würde man mit dieser Vorgehensweise sicher zum Ziel kommen. Die ersten Rechenmaschinen haben tatsächlich auf diese Weise multipliziert. Dieses Verfahren ist jedoch mühsam und zeitaufwendig. Deshalb wurde eine bessere Methode entwickelt.

Das Multiplizieren zweier Zahlen schreibt man üblicherweise so:

```
 14 x 12
 14
+ 28
 168
```

(Oft wird eine freie Stelle auch mit einer 0 aufgefüllt.)



In der Darstellung wird 100101 (37) mit 1111 (15) multipliziert – die Zahl 100101 erscheint viermal als „Kopie", wobei jede „Kopie" entsprechend der Position der multiplizierenden 1 nach rechts verschoben ist. Zum Schluß werden alle „Kopien" addiert und ergeben als Resultat 1000101011, dezimal die Zahl 555.

Genau dieses Verfahren wird auch im binären Zahlensystem angewendet:

```
 101 x 11
 101
+ 101
 1111
```

Dies gilt auch für größere Zahlen. Das Beispiel 14 x 12 sieht binär so aus:

```
 1110 x 1100   (14 x 12)
 1110
  1110
   0000
+  0000
 10101000      (168)
```

Multipliziert man eine Zahl mit 1, bleibt die Zahl unverändert (14 x 1 = 14), multipliziert man mit 0, ist das Ergebnis 0 (14 x 0 = 0). Dies gilt in allen Zahlensystemen.

Das binäre Multiplizieren besteht aus drei schematischen Vorgängen: Kopieren, Verschieben (shifting) und Addieren. Genau dies ist auch die Arbeitsweise des Computers. Zuerst wird die zu multiplizierende Zahl entsprechend der Anzahl der Einsen im Multiplikator kopiert. Steht im Multiplikator eine 0, entsteht anstelle einer Kopie eine Reihe von Nullen. Im nächsten Schritt werden Kopien und Null-Reihen untereinander gesetzt und entsprechend der Stellen des Multiplikators verschoben. Dann werden die Zahlen spaltenweise addiert.

Ein Computer braucht zum Multiplizieren ein riesiges Aufnahmevermögen für die Stellen. Allgemein kann gesagt werden, daß das Ergebnis einer Multiplikation die doppelte Stellenzahl der größeren Zahl haben kann.

Multiplikationsergebnisse von Computern können aber auch falsch sein – eine Feststellung, die manchen überraschen mag. Die Ursache dafür ist in fast allen Fällen auf eine unzureichende Stellenzahl für die Ergebnisspeicherung zurückzuführen. Sie wird von der Hardware bestimmt. Bei zu wenigen Stellen tritt ein sogenannter „Überlauf" (overflow) auf. Die letzten Stellen gehen verloren, und das Ergebnis wird falsch.

Gesetze der Logik

Schon ein Jahrhundert vor der Erfindung des Computers veröffentlichte George Boole die von ihm gefundenen Regeln der mathematischen Logik.

George Boole, 1815 in Lincoln als Sohn eines Schusters geboren, glaubte daran, menschliches Denken auf mathematische Weise beschreiben zu können. Er brachte sich selbst Mathematik bei und begann die Erforschung der Logik menschlicher Entscheidungen. Ohne seine revolutionären Ideen wäre die Entwicklung der Computer von heute nicht möglich gewesen. George Boole starb jedoch schon 1864 – ein Jahrhundert, bevor die rasante Entwicklung der Computertechnologie begann.

Das Beispiel einer Tanzveranstaltung soll die Denkweise des Mathematikers verdeutlichen. Angenommen, Sie befinden sich in einem Saal und suchen einen Tanzpartner. Einige Paare tanzen, andere Menschen stehen etwas abseits. Logischerweise tut niemand beides gleichzeitig. Ihr auszuwählender Tanzpartner ist entweder männlich oder weiblich, aber nicht beides.

Boole würde die Situation folgendermaßen analysiert haben. Er hätte die Anwesenden vier Gruppen zugeordnet: männliche Personen (m), weibliche Personen (w) und Personen, die gerade tanzen (t) bzw. nicht tanzen (n).

Informationsstrukturen

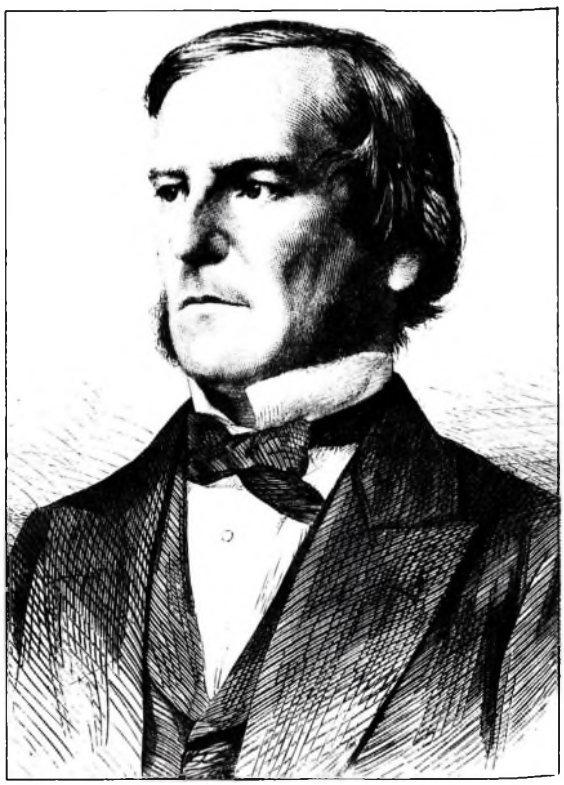
Der Tanzpartner für einen Mann müßte zwei Bedingungen erfüllen: Er müßte weiblich sein und in diesem Augenblick gerade nicht tanzen. Boole erkannte, daß das 'und' bei der Verbindung dieser beiden Bedingungen eine wichtige Funktion erfüllt. Er bezeichnete sie mit einem umgedrehten U. Die Gruppe der möglichen Tanzpartner läßt sich also als w∩m definieren.

Wenn der Mann nicht tanzen, sondern lediglich eine Unterhaltung führen möchte, so könnte er jeden aus den Gruppen m oder w auswählen, da diese alle im Raum befindlichen Menschen umfassen. Die Funktion ‚oder' bezeichnete Boole mit dem Symbol U. Dieser Logik entsprechend beinhaltet mUw also alle Männer und Frauen im Raum. Nach diesen Booleschen Symbolen sind die logischen Gatter, mit denen Computer arbeiten, benannt worden.

Nun zu einer interessanten Anwendung der Boolschen Logik. Die englischen Mathematiker John Venn (1834–1923) und Charles Dodgson (1832–1892), besser bekannt als der Schriftsteller Lewis Caroll, entwickelten sie. Als Beispiel soll ein praktisches Problem dienen. Angenommen, Sie haben eine Liste mit den Namen ihrer Freunde und Bekannten im Heimcomputer gespeichert. Zu jedem Namen haben Sie beispielsweise noch die Telefonnummer und das jeweilige Hobby aufgelistet. Eines Nachmittags möchten Sie mit einem Freund oder einer Freundin Tennis auf einem Platz am anderen Ende der Stadt spielen. Ihr

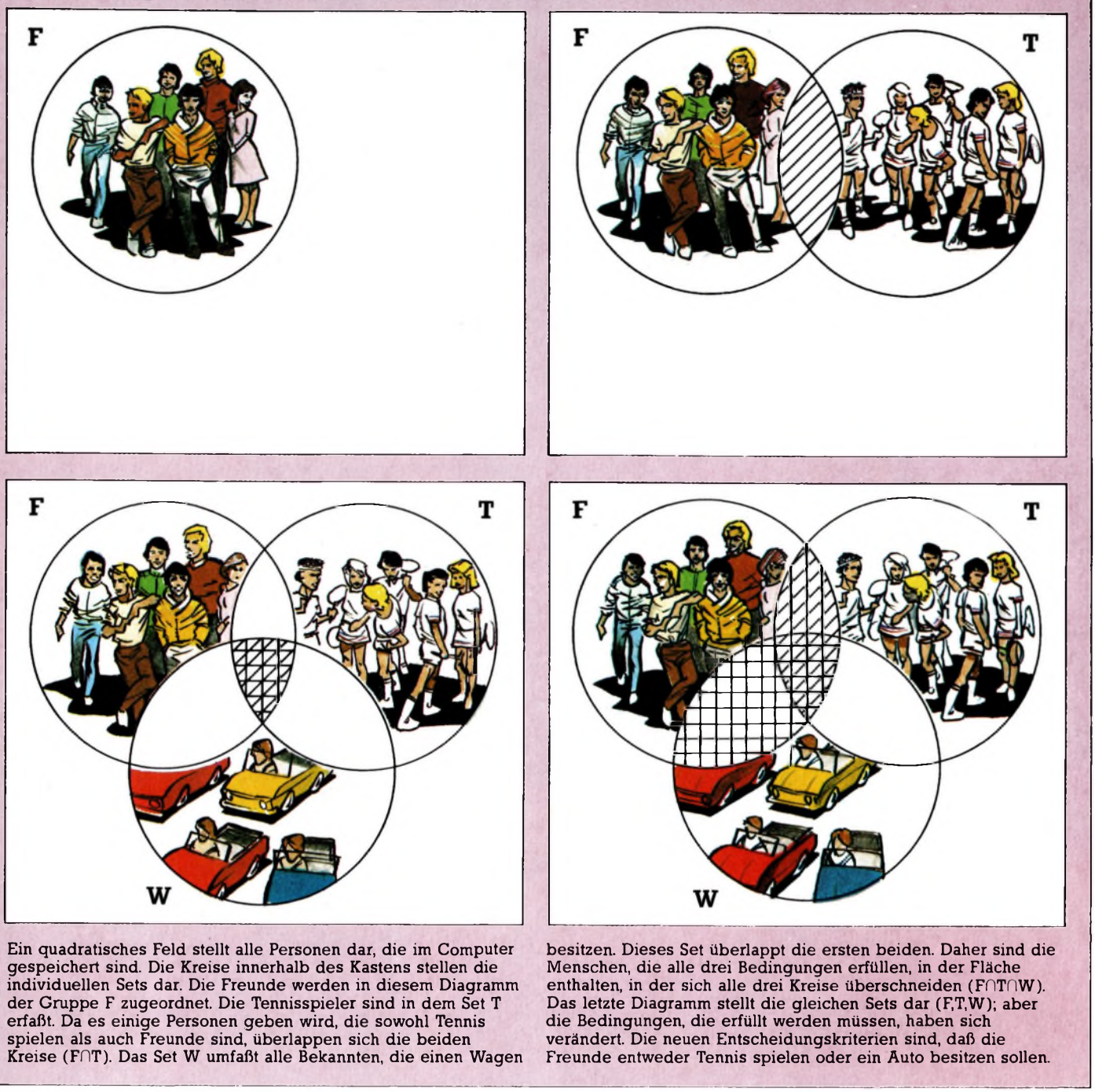
Ein Programm für Freunde

```
10 DIM N$(10), D$(10), F$(10), T$(10), C$(10)
15 REM NAME, TEL.NR., FREUND?, TENNIS?, AUTO?
17 PRINT "GIB DIE DATEN IN DIESER FORM EIN:"
18 PRINT "NAME, TELEFON, JA/NEIN, JA/NEIN, JA/NEIN"
20 FOR K=1 TO 10
30 INPUT N$(K), D$(K), F$(K), T$(K), C$(K)
40 NEXT K
45 REM SUCHROUTINE
50 FOR J=1 TO 10
60 IF F$(J)="JA" AND T$(J)="JA" AND C$(J)="JA" THEN GOSUB 100
70 NEXT J
80 END
100 PRINT N$(J), D$(J)
110 RETURN
```



George Boole lebte in einer Zeit, als man sich Computer noch nicht einmal vorstellen konnte und war doch ein Begründer der mathematischen Logik, die den heutigen Computern zugrunde liegt. 1847 veröffentlichte er seine Erkenntnisse und wurde dadurch fast über Nacht berühmt. Bald wurde er zum Professor für Mathematik an der neugegründeten Universität von Cork.

Venn-Diagramme



Ein quadratisches Feld stellt alle Personen dar, die im Computer gespeichert sind. Die Kreise innerhalb des Kastens stellen die individuellen Sets dar. Die Freunde werden in diesem Diagramm der Gruppe F zugeordnet. Die Tennisspieler sind in dem Set T erfaßt. Da es einige Personen geben wird, die sowohl Tennis spielen als auch Freunde sind, überlappen sich die beiden Kreise (F∩T). Das Set W umfaßt alle Bekannten, die einen Wagen besitzen. Dieses Set überlappt die ersten beiden. Daher sind die Menschen, die alle drei Bedingungen erfüllen, in der Fläche enthalten, in der sich alle drei Kreise überschneiden (F∩T∩W). Das letzte Diagramm stellt die gleichen Sets dar (F,T,W); aber die Bedingungen, die erfüllt werden müssen, haben sich verändert. Die neuen Entscheidungskriterien sind, daß die Freunde entweder Tennis spielen oder ein Auto besitzen sollen.

Computer soll die Namen und Telefonnummern aller Personen ausdrucken, die gern Tennis spielen, gute Freunde sind und ein Auto haben (damit sie gemeinsam zum Platz fahren können). Deshalb untersucht das ‚Programm für Freunde' (siehe Kasten) zunächst folgende Informationen: Ist es ein Freund/eine Freundin? Hat er/sie einen Wagen? Kann er/sie Tennis spielen?

In dem Programm ist die Anzahl der gespeicherten Personen auf zehn festgelegt. Diese Zahl können Sie selbstverständlich auch ändern. Denken Sie jedoch daran, auch die 10 in der Klammer hinter den DIM-Anweisungen in Zeile 10 zu ändern. Die Liste wird dann mit einem IF-THEN-Befehl durchsucht, mit dem eine Anzahl von Bedingungen abgefragt werden. Die meisten BASIC-Versionen bieten die Möglichkeit, in dem IF-THEN-Befehl eine ganze Anzahl von Bedingungen mit den logischen Operatoren AND und OR miteinander zu verbinden.

In manchen Programmen gibt es sehr komplexe Kombinationen der logischen Funktionen. Die Bedeutung von Booles Algebra, die zu dessen Lebzeiten nur wenig mehr als ein Kuriosum darstellte, ist erst heute, im Computerzeitalter, erkannt worden.

Richtig adressiert

Die Zentraleinheit CPU ordnet Transport und Manipulation von Daten und Befehlen, die in Tausenden von Speicherplätzen des Computers abgelegt sind.

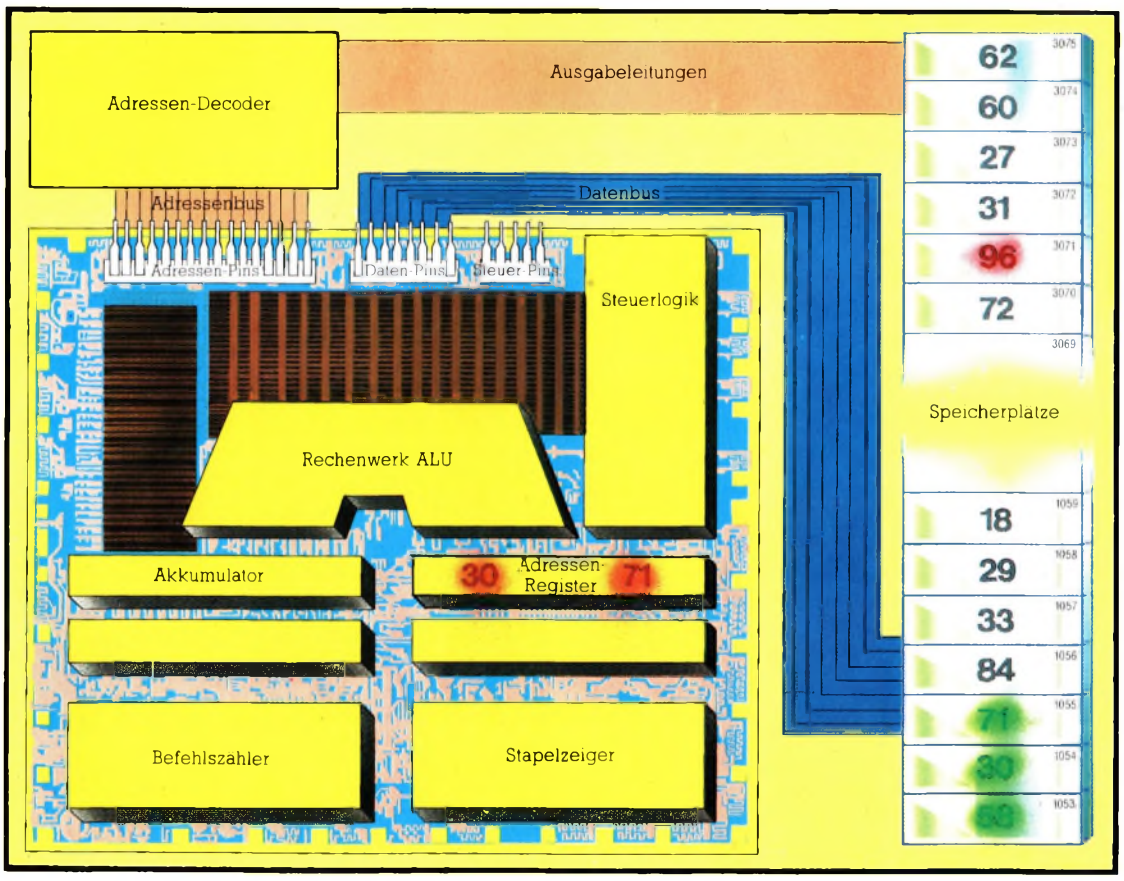
Die CPU verschafft sich durch Adressierung des gewünschten Speicherplatzes und Laden seines Inhaltes über den Datenbus Zugang zu gespeicherten Befehlen oder Daten. Dieser Vorgang ist reichlich kompliziert, weil sich hinter den gespeicherten Bytes (8 Bit umfassende binäre Codes) sowohl Daten, die vom Prozessor nach unterschiedlichen Vorgaben manipuliert werden müssen, als auch Befehle verbergen können.

Die vielen Tausend Speicherplätze des Computerspeichers enthalten schließlich beides: Befehle, die der CPU sagen, was zu tun ist, und Daten, an denen sie diese Manipulationen auszuführen hat. Wie aber erkennt die Zentraleinheit den Unterschied zwischen Befehlen und Daten?

Befehle sind codierte binäre Ausdrücke, die die CPU anweisen, bestimmte Operationen in vorgegebener Reihenfolge auszuführen. Erkennt die CPU im binären Ausdruck 00111010 z. B. einen Befehl und nicht nur ein Datenbyte, so könnte sie die nächsten beiden Speicherplätze adressieren, deren Inhalt in ein besonderes Adressen-Register laden, die Adressenleitungen dem Inhalt dieses Registers entsprechend setzen und dann den Inhalt der neu adressierten Speicherplätze über den Datenbus in den Akkumulator laden.

Speicherplatz-Adresse

Dies hört sich zwar etwas verwirrend an, doch wir haben gerade eine Adressierung von Speicherplätzen beschrieben, wie sie in der weit verbreiteten CPU Z80 vorkommt. Die Illustration verdeutlicht dies (alle verwendeten Zahlen sind Dezimalzahlen). Nehmen wir also an, die CPU weiß, daß das nächste Byte mit Adresse 1053 kein Datenbyte sondern ein Befehl ist. Dann wird sie über den Adressenbus 1053 oder binär 0000010000011101 senden, und ihre Adressenleitungen dem binären Ausdruck entsprechend auf Null oder Eins setzen.



Selbst die einfachste CPU besteht aus vielen Funktionsblöcken. Befehle, im Englischen kurz ‚opcodes' genannt, müssen vom Speicher in die CPU geladen werden. Nach der Decodierung durch die Steuerlogik werden die Befehle ausgeführt. Im illustrierten Beispiel löst der vom Speicherplatz 1053 geladene Befehl 58 eine Kette von Aktionen aus: Der Inhalt von Speicherplatz 1054 wird in eine Hälfte des 16-Bit-Adressen-Registers geladen, der Inhalt von 1055 in die zweite Hälfte. Nachdem beide Bytes dieses Registers auf den Adressenbus gegeben und decodiert sind, wird 96, der Inhalt des so adressierten Speicherplatzes 3071, von der CPU über den Datenbus in den Akkumulator geladen. Dann wird der Adressenbus auf 1056 gesetzt. Das ist die Adresse vor dem Sprung nach 3071, aber um Eins erhöht. Durch die bei der Konstruktion des Chips festgelegten Verkettungen von Daten und Befehlen erkennt die CPU den Inhalt als Befehl. In unserem Beispiel bewirkt 84 deshalb eine Invertierung des Akkumulator-Inhaltes.

Der Adressen-Decoder ändert daraufhin nur eine Ausgabe-Leitung, die Speicherplatz 1053 adressiert.

Als nächstes wird der adressierte Inhalt, 58 oder 00111010 binär, über den Datenbus in die CPU geladen. Da die CPU einen Befehl erwartet, bestimmt die Steuerlogik, welche Operationen ablaufen sollen. In unserem Beispiel hat der Befehl folgenden Inhalt: Benutze die nächsten 16 Bit als Adresse und lade den Inhalt des adressierten Speicherplatzes in den Akkumulator. Sobald die CPU diesen Befehl über die Steuerlogik erkennt, weiß sie, daß die nächsten beiden Bytes eine Adresse sein müssen, deren Inhalt sie in den Akkumulator laden muß. Daraus folgt, daß die CPU erst nach Ausführung dieser Operationen den nächsten Befehl erwarten kann und der nächste Befehl vom Speicherplatz 1056 kommen muß.

Die CPU ändert in unserem Beispiel also zuerst den Adressenbus von 1053 auf 1054 und lädt den Inhalt dieses Speicherplatzes über den Datenbus in eine Hälfte des Adressen-Registers. Dann ändert sie den Adressenbus auf 1055 und lädt das so adressierte Byte in die zweite Hälfte des Adressen-Registers. Der Transport der Daten zur CPU geschieht also immer über den Datenbus, einer Art Einbahnstraße, die von den Speicherplätzen zur CPU führt.

Die CPU überträgt nun den Inhalt des Adressen-Registers auf den Adressenbus: Die neue Adresse lautet dann 3071 oder binär 0000101111111111. Nach Aktivierung des Speicherplatzes 3071 durch den Adressen-Decoder lädt die CPU 96 oder binär 01100000 über den Datenbus in den Akkumulator und gibt 1056, die um Eins erhöhte Adresse vor dem Sprung nach 3071, auf den Adressenbus. Die CPU erwartet unter dieser Adresse einen Befehl. Erinnern Sie sich bitte, daß der bis jetzt geschilderte Vorgang durch **einen** Befehl ausgelöst wurde und automatisch abläuft.

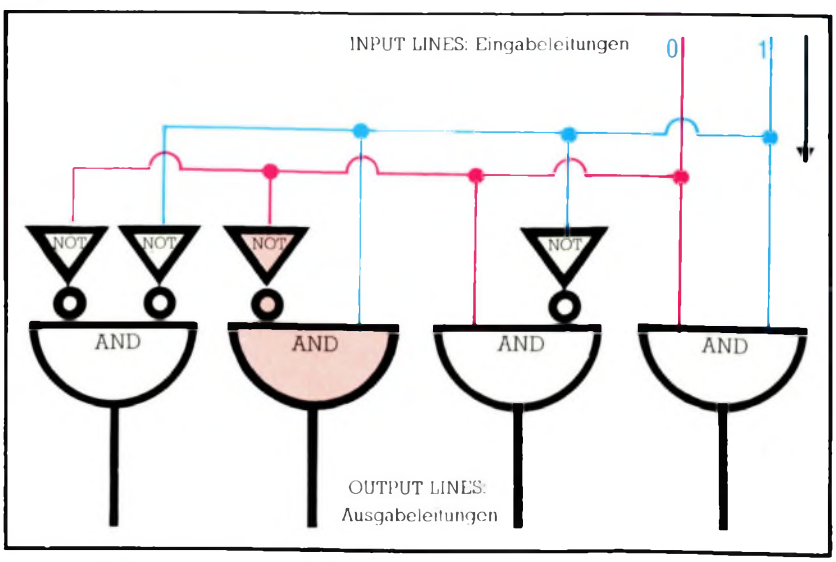
CPU-Befehlsvorrat

Welcher Art könnte der unter Adresse 1056 gespeicherte Befehl sein? Die Antwort ist nicht einfach, weil der Befehlsvorrat der CPU, abhängig vom Typ, mehrere Dutzend oder einige Hundert umfassen kann. Nehmen wir an, daß die Daten im Akkumulator invertiert werden sollen (Nullen werden in Einsen, Einsen in Nullen verwandelt). Nehmen wir weiter an, der Inhalt des Speicherplatzes 1056 wäre ein Befehl, der diese Operation ausführt, und hieße 84. Die CPU wird dann, sobald sie 84 über den Datenbus geladen hat, den Akkumulatorinhalt 96 oder binär 01100000 in 159 oder binär 10011111 verwandeln. Nach diesem Befehl zur Invertierung erwartet die CPU einen weiteren Befehl, denn im Akkumulator befindet sich ja noch das invertierte Datenbyte.

Die beschriebene Art, Speicherplätze zu adressieren, ist nur eine von vielen Möglichkeiten, die der Programmierer zur Verfügung hat. Auch sind die verwendeten Befehle, 58 für das Laden des Akkumulators und 84 für das Invertieren seines Inhalts, nur für die modellhafte CPU unseres Beispiels gedacht. Das vorgeführte Prinzip gilt jedoch generell, auch wenn die verschiedenen Hersteller vom Microprozessoren gleiche Befehle unterschiedlich codieren und die Befehlsätze nicht immer gleichen Umfang haben.

Für die Eingabe/Ausgabe (E/A) von Daten müssen ebenfalls feste Speicherplätze reserviert werden, die von der CPU genau wie Daten oder Befehle adressiert werden. Gewöhnlich hat ein 8-Bit-Microprozessor nur acht Leitungen (Pins) für die Adressierung von E/A-Speicherplätzen. Dies bedeutet, daß maximal nur 256 Eingabe/Ausgabe-Adressen belegt werden können. Für die meisten der möglichen Anwendungen von Microcomputern ist dies jedoch mehr als genug.

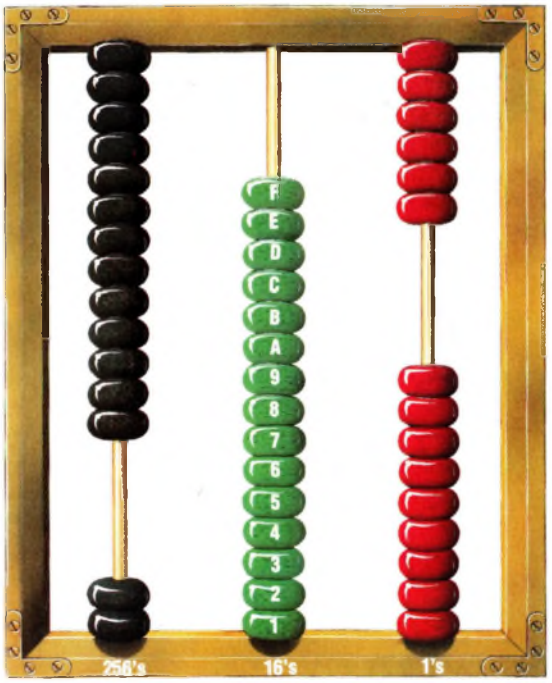
Bei der Adressierung ist besonders darauf zu achten, daß nur die Adresse angesprochen wird (entweder eine Speicherplatz- oder eine Ein/Ausgabeadresse), die beim jeweiligen Programmablauf benötigt wird. Die anderen Adressen dürfen dabei nicht aktiviert werden. Jede Adressierung erfordert auch die Fähigkeit, Adressen zu decodieren, damit nur der eine gewünschte Speicherplatz freigegeben wird. Diese Art der Freigabe ist mit einfachen Logikbausteinen zu bewältigen, wenn nur wenige Adressenleitungen zu decodieren sind. Das Prinzip eines 2-zu-4-Decoders zeigt die Illustration. Häufig geschieht die Auswahl von E/A-Geräten auf diese einfache Weise. Mit steigender Anzahl von Adressenleitungen steigt jedoch der Aufwand für die Decodierung überproportional. Muß beispielsweise von 65 536 verschiedenen Speicherplätzen jeder für sich eindeutig adressierbar sein, so geschieht dies meistens direkt im Speicherbaustein selbst.



Der 16 Leitungen umfassende Adressenbus kann jeden einzelnen von 65 535 möglichen Speicherplätzen eindeutig adressieren. Adressen-Decoder setzen dazu die Nullen und Einsen auf dem Adressenbus in ein Signal um, das nur einen Speicherplatz freigibt. Der größte Teil der Decodierung wird durch logische Schaltungen vorgenommen, die sich auf dem Speicherchip befinden. Die Illustration zeigt, wie mit zwei Eingabeleitungen immer nur eine von vier möglichen Ausgabeleitungen ausgewählt werden kann.

Rechnen mit Basis 16

Programmierer benutzen hexadezimale Zahlen, obwohl Computer nur binäre Zahlen, also nur zwei logische Zustände, erkennen und verarbeiten können; denn Hexzahlen lassen sich sehr einfach in Binärzahlen umrechnen.



Ein Abakus für hexadezimale Zahlen müßte auf jeder Stange 15 Perlen haben. Das hexadezimale Zahlensystem hat die Basis 16 und fünfzehn Ziffern: von 0 bis 9 und A bis F. Die mittlere Stange des Abakus läßt erkennen, daß A für dezimal 10 steht, B für 11, C für 12, D für 13, E für 14 und F für 15. Hat die Zählung der Einer-Perlen F erreicht, wird die erste Sechszehner-Perle gezählt und die Zählung der Einer-Perlen beginnt von vorn, bis alle Sechszehner-Perlen gezählt sind. Der abgebildete Abakus stellt die Dezimalzahl 761 dar. Sie wird gebildet aus 2×256 + 15×16 + 9×1. Ihre hexadezimale Form ist 2F9.

Es ist offensichtlich, warum für Computer das binäre Zahlensystem so vorteilhaft ist: Ein/Aus-Zustände elektrischer Signale lassen sich als Einsen und Nullen darstellen oder codieren. Wir haben uns dagegen an das Dezimalsystem mit der Basis Zehn gewöhnt, und schon Kleinkinder können leichte Rechenaufgaben lösen, weil sie ihre zehn Finger zu Hilfe nehmen können.

Warum nun aber gerade hexadezimale Zahlen mit der Basis 16? Nun, sie lassen sich viel leichter als Dezimalzahlen in binäre Zahlen verwandeln, und umgekehrt gilt das gleiche. Trotzdem stellt sich die Frage, warum man Programme nicht gleich in binärer Form schreibt, wenn Computer binäre Zahlen so gut verstehen. Zwei Gründe sind dafür ausschlaggebend: Erstens sind binäre Ausdrücke viel umfangreicher als ihre dezimalen oder hexadezimalen Äquivalente, und zweitens macht man, wegen der vielen Einsen und Nullen, sehr häufig Fehler. Nehmen wir als Beispiel 356 dezimal. Ihre vielstellige binäre Form, 101100100, ist viel weniger anschaulich.

Das hexadezimale System arbeitet mit 16 verschiedenen Ziffern einschließlich der Null. Die größte einstellige hexadezimale Zahl entspricht dezimal 15. Da mit diesem System nur zehn verschiedene Ziffern darstellbar sind, nimmt man für die hexadezimalen Zahlen, die dezimal 10 bis 15 entsprechen, die ersten sechs Buchstaben des Alphabets zu Hilfe. Wir zählen hexadezimal also wie gewöhnlich von 0 bis 9 und dann A, B, C, D, E, F. F entspricht also dezimal 15. Eine mehrstellige hexadezimale Zahl kann demnach aus Ziffern und Buchstaben bestehen. Die Tabelle zeigt Dezimalzahlen bis 21 und ihre binäre und hexadezimale Schreibweise. Hex 15 besteht aus 5 Einern und einem Sechszehner, was zusammen in dezimaler Schreibweise 21 ergibt.

FFFF ist die hexadezimale Schreibweise für 65 535. Der Vorteil liegt auf der Hand: Die Dezimalzahl hat fünf Stellen, die entsprechende Binärzahl 16 Stellen, und die gleichwertige „Hexzahl" hat nur vier Stellen. Ein zusätzlicher Vorteil ist, daß eine hexadezimale Stelle oder Ziffer genau vier binäre Stellen wiedergeben kann. Dies vereinfacht, wie wir gleich sehen werden, die Umwandlung von binär in hexadezimal und zurück.

Wir beginnen von rechts und teilen die binäre Zahl in Gruppen zu je vier Ziffern. Für jede binäre Vierergruppe ist einfach die entsprechende hexadezimale Ziffer einzutragen. Hier sind einige Beispiele:

		1110	1001	binär
		=E	9	hex
		=233		dezimal
	1111	1000	1100	binär
	=F	8	C	hex
	=3980			dezimal
0111	1110	0010	1101	binär
=7	E	2	D	hex
=32301				dezimal

Wofür benötigt man nun eigentlich hexadezimale Zahlen? Denn um Programme in BASIC zu schreiben, sind weder hexadezimale noch binäre Zahlen nötig. Richtig, aber wer Programme in Maschinencode oder ASSEMBLER schreiben will, tut dies sehr bequem mit hexadezimalen Zahlen.

Zur Unterscheidung von Dezimalzahlen schreibt man hinter hexadezimalen Zahlen ein H. Dezimal 256 erscheint dann als 100H und wird gelesen: Eins, Null, Null, Hex.

Binär	Dezimal	Hex
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
00000101	5	5
00000110	6	6
00000111	7	7
00001000	8	8
00001001	9	9
00001010	10	A
00001011	11	B
00001100	12	C
00001101	13	D
00001110	14	E
00001111	15	F
00010000	16	10
00010001	17	11
00010010	18	12
00010011	19	13
00010100	20	14
00010101	21	15

Detektivarbeit

Bei der Datenübertragung von einem Rechner zum anderen besteht die Gefahr, daß sich Fehler einschleichen. Sie können mit Hilfe von ‚Hamming-Codes' entdeckt und beseitigt werden.

Geschichten von spektakulären Computerfehlern, z. B. von dem Computer, der ein und denselben Brief 500mal an den Empfänger geschickt hat, sind hinlänglich bekannt. Der Grund liegt hier natürlich nicht im Gerät, sondern in menschlichem Fehlverhalten. Der Computer hat lediglich die Wirkung eines Verstärkers – aus einem simplen Tippfehler wird dann der berühmte „Elefant".

Es gibt allerdings auch Computerfehler, die nicht dem Menschen zuzuschreiben sind – allgemein als „Bit-Fehler" bekannt. Ein solcher Fehler entsteht, wenn ein einzelnes Bit innerhalb einer Datenfolge „umkippt", also von 1 zu 0 wird oder umgekehrt. Die Ursache kann in fehlerhafter Hardware liegen, z. B. in einem fehlerhaften RAM-Chip. Dies ist auch der Grund, warum viele Heimcomputer erst einmal eine „Diagnose"-Routine durchlaufen, sobald sie eingeschaltet werden.

Die meisten Bitfehler jedoch sind nicht hardwarebedingt. Eine sommerliche Hitzewelle kann dazu führen, daß irgendwo im Computer ein Hitzestau entsteht. Er verursacht zwar kaum einen dauerhaften Schaden, kann aber der Grund dafür sein, daß aus einem A auf dem Bildschirm plötzlich ein B wird. Ist das fehlerhafte Bit Teil eines wichtigen Zeigers, kann dies allerdings dazu führen, daß das Programm „abstürzt" und der Computer neu aktiviert werden muß.

Auch Sonnenflecken können Bitfehler verursachen, wenn subatomare Partikel in die Atmosphäre eindringen und den Elektronenfluß in den Schaltkreisen des Rechners überlagern. In militärischen Systemen, industriellen Steuerungen, wissenschaftlichen Experimenten oder im internationalen Bankwesen könnten Bitfehler katastrophale Folgen haben. Deshalb wurden viele Methoden zum Aufspüren von Bitfehlern entwickelt. Die einfachste ist die Paritätsprüfung (beschrieben im folgenden Artikel). Eine Alternative dazu ist die ‚Kontrollsumme'. Von dieser Methode wird weitgehend Gebrauch gemacht, wenn Daten auf Magnetband oder Diskette geschrieben werden. Die Kontrollsumme steht am Ende eines Datenblocks. Ein Datenblock besteht aus 128 Bytes; dabei hat jedes Byte einen Wert zwischen 0 und 255. Die gespeicherten Daten werden addiert und durch 256 geteilt. Der bei dieser Rechnung verbleibende „Rest" stellt die ‚Kontrollsumme' dar. Ein Beispiel:

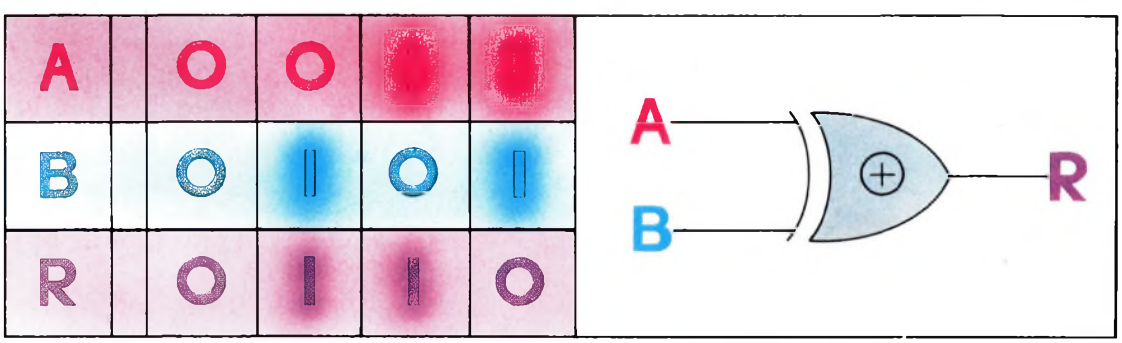
Daten 114, 67, 83 . . . (121 weitere Werte)
. . . 36, 154, 198
Gesamtsumme der 137 Bytes = 16 673
Dividiert durch 256 = 65, Rest 33
Kontrollsumme daher = 33

Die Kontrollsumme wird als 128. Byte geschrieben. Wenn der Rechner einen Datenblock liest, führt er eine eigene Kontrollsummenrechnung durch und vergleicht das Ergebnis mit der vom letzten Byte repräsentierten Kontrollsumme. Differieren die beiden Zahlen, weiß der Rechner, daß sich in die Übertragung ein Bitfehler eingeschlichen hat.

Beide Methoden sagen dem Computer nicht, welches Bit fehlerhaft ist. Taucht der Fehler nur beim Übertragen auf, kann der empfangende Computer veranlassen, daß ein bestimmtes Byte oder ein ganzer Byte-Block noch einmal übertragen wird.

Der Hamming-Code

Für die Anwendungen, bei denen unter keinen Umständen Fehler auftreten dürfen, mußte eine Methode gefunden werden, die Fehler sowohl entdeckt als auch korrigiert. Eine solche Methode sind die sogenannten ‚Hamming-



Ein einfaches Exklusiv-Oder-Gatter (kurz Ex-Oder) hat zwei Eingänge und einen Ausgang. Liegt auf beiden Eingängen ein 0-Signal, erscheint am Ausgang ebenfalls ein 0-Signal. Liegt an einem der beiden Eingänge ein 1-Signal, ist auch der Ausgang 1, aber nicht, wenn an beiden Eingängen zugleich 1 anliegt. Dieser letzte Zustand unterscheidet das Ex-Oder vom Oder-Gatter. Der Vorgang läßt sich auch durch eine Wertetabelle darstellen. Ein Ex-Oder mit mehr als zwei Eingängen ergibt ein 1-Signal als Ausgang, wenn eingangsseitig eine ungerade Anzahl von 1-Signalen anliegt. Gatter dieser Art werden dazu verwendet, Paritäts- oder Prüfbits zu erzeugen.

Codes', benannt nach ihrem Erfinder R. W. Hamming von der amerikanischen Firma Bell Telephone Laboratories.

Jede Fehlerkonstruktion arbeitet nach dem Prinzip der Redundanz (was soviel wie „Überfluß" bedeutet). Die menschliche Sprache enthält einen hohen Grad an überflüssigen, für die Information nicht wichtigen Elementen. In Schriftstücken lassen sich unleserliche Passagen oft aus dem inhaltlichen Zusammenhang heraus rekonstruieren. Im Funkverkehr werden statt der Buchstaben A, B, C usw. die Worte Alpha, Bravo und Charlie benutzt – eine spezielle Redundanz also –, um Verständigungsfehlern vorzubeugen.

Nehmen wir als Beispiel ein Wort von x Bit Länge. Es soll aus y Datenbits und z Redundanzbits bestehen ($x = y + z$). Hamming-Codes erfordern einen hohen Wert für z. Man geht davon aus, daß ein Bitfehler in jedem Bit auftreten kann, auch in einem Redundanzbit. Ein Fehler tritt in einem Wort mit der Wahrscheinlichkeit von eins zu einer Million auf. Beim Empfang der Daten am anderen Übertragungsende bestehen folglich $x + 1$ Möglichkeiten. Entweder gibt es überhaupt keine Fehler oder das erste Datenbit ist fehlerhaft und so weiter bis zum x-ten Bit. Mit z Redundanzbits können 2^z Möglichkeiten repräsentiert werden, so daß für ein Wort, das die Prüfung auf einen Bitfehler bestehen soll, gilt:

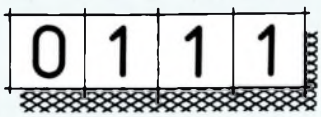
$$2^z = y + z + 1$$

Ist $y = 7$ (bei ASCII-Codes), muß $z = 4$ sein. Ist $y = 4$, muß $z = 3$ sein. Bei $y = 16$ jedoch muß der Wert für z nur auf 5 erhöht werden. Hamming-Codes sind also für längere Wörter sehr viel besser geeignet als für kürzere.

Im Hamming-Code wirkt jedes Redundanzbit als Prüfbit für ‚gerade Parität', und zwar abhängig von der Bitkombination im Wort. Tritt bei der Datenübertragung ein Bitfehler auf, wird eines (oder werden mehrere) der Prüfbits unrichtig sein und durch seine (ihre) Stellung innerhalb der Bitkombination auf das fehlerhafte Bit hinweisen. Der empfangende Computer ist damit in der Lage, das „entdeckte" Fehlerbit zurückzuweisen.

Der Schlüssel zur Funktionsweise der Hamming-Codes liegt in der unterschiedlichen Kombination der zu prüfenden Bits. Die Gesamtzahl der Bits wird in unterschiedliche, sich überlappende Sätze aufgeteilt, und zwar so, daß sich zwei Bits nicht in der gleichen Kombination wiederholen. Der empfangende Computer prüft die gleichen Sätze auf Parität, die der sendende Computer prüfte, um den Hamming-Code zu erzeugen. Kippt beim Übertragen eines der Bits (einschließlich der Hamming-Bits), wird einer oder werden mehrere dieser Sätze die Paritätsprüfung nicht bestehen. Aus der Kombination der nicht bestandenen Paritätsprüfungen ergibt sich dann der Hinweis auf das fehlerhafte Bit.

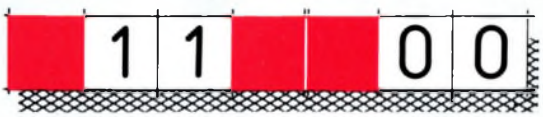
Wie der Hamming-Code arbeitet



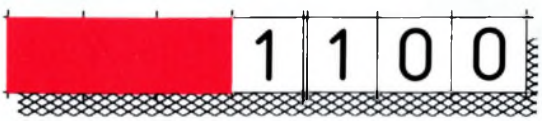
Diese vier Bits sollen übertragen werden.

Der Computer fügt drei weitere Bits als Hamming-Code hinzu. Sie sind nach einem bestimmten Muster angeordnet:

In diesem Vierermuster muß die 1 in gerader Anzahl vorhanden sein.



Auch in dieser Anordnung muß die 1 in gerader Anzahl erscheinen.

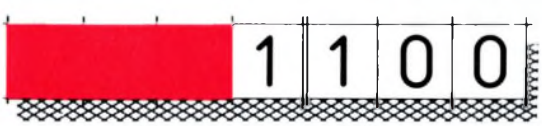


Eine gerade Anzahl von 1-Signalen muß auch dieses Muster zeigen. Der Computer muß drei Gleichungen hintereinander lösen.

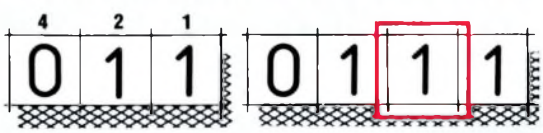
In diesem Muster wird unterstellt, daß das dritte Bit von links von 1 auf 0 gekippt ist.

Führt der Computer nun die erste der drei Prüfungen durch, wird diese nicht bestanden, weil die 1 in ungerader Anzahl erscheint. Das zeigt zwar einen Fehler, nicht aber das fehlerhafte Bit.

Auch die zweite Prüfung wird nicht bestanden.



Die dritte Prüfung jedoch wird bestanden, weil die 1 in gerader Anzahl erscheint.



So entsteht der Hinweis auf das fehlerhafte Bit. Bezeichnet man eine nicht bestandene Prüfung mit 1 und eine bestandene mit 0 und schreibt die Ergebnisse in umgekehrter Reihenfolge auf, erhält man eine binäre 3 als Hinweis darauf, daß das dritte Bit das Fehlerbit ist und von 0 auf 1 zurückgestellt werden muß.

Die Methode ist auch wirksam, wenn der Fehler bei einem Hamming-Bit auftritt. Schlagen alle drei Prüfungen fehl, erhält man binär 111, was anzeigt, daß das äußerste rechte Bit fehlerhaft ist. Werden alle drei Prüfungen bestanden, ist die Übertragung fehlerfrei.

Auf Fehlersuche

‚Gerade Parität' stellt sicher, daß die Anzahl der 1-Bits in einem Byte immer gerade ist. Dies erleichtert das Erkennen von Übertragungsfehlern, hilft aber nicht bei der Suche nach der Position des ‚Bitfehlers'.

Digitalcomputer zeichnen sich gegenüber Analogrechnern dadurch aus, daß sich Fehler oder Ungenauigkeiten, wie sie in elektrischen Stromkreisen auftreten können, beim Durchlaufen der Stromkreise nicht „aufschaukeln". Bei der Fernübertragung von Daten, sei es über eine serielle Schnittstelle und zwei Drähte oder über die Fernsprechleitung, kann das elektrische „Hintergrundrauschen" in der Leitung jedoch dazu führen, daß ein Bit von 1 auf 0 oder umgekehrt kippt. Normalerweise hat der empfangende Computer keine Möglichkeit, Übertragungsfehler zu erkennen. Er wird fehlerhafte Daten als korrekt annehmen.

Wie sich ein Bitfehler auswirken kann, zeigt nachfolgendes Beispiel:

Übertragen wird [] 1010001 (ASCII-Code für Q).

Empfangen wird [] 1000001 (ASCII-Code für A).

Ein Übertragungsfehler dieser Art ist im besten Fall „nur" ein Ärgernis, kann aber auch katastrophale Folgen haben. ASCII-Codes arbeiten mit Werten bis zu 127, für die nur sieben Bits benötigt werden. Das achte Bit steht als ‚Paritätsbit' zur Verfügung. Dieses MSB (Most Significant Bit) prüft, ob ein Übertragungsfehler vorliegt. Es steht in der hier verwendeten Schreibweise in eckigen Klammern vor den eigentlichen Datenbits.

Das Paritätsbit kann auf „gerade" oder „ungerade" Parität prüfen. Hier geht es um die gerade Parität. Sie ist gegeben, wenn das Paritätsbit so eingestellt ist, daß die Gesamtzahl der 1-Bits innerhalb eines Bytes immer gerade ist. Die ASCII-Codes für A und Q sehen bei gerader Parität so aus:

[0]1000001 für A
[1]1010001 für Q.

Der A-Code enthält zwei 1-Bits; wird das Paritätsbit, auch Prüfbit genannt, als 0 gesetzt, ist die Gesamtzahl aller 1-Bits gerade. Im Q-Code dagegen finden sich drei 1-Bits; um eine gerade Anzahl von 1-Bits zu erhalten, wird das Prüfbit als 1 gesetzt, so daß dieses Byte jetzt vier 1-Bits enthält.

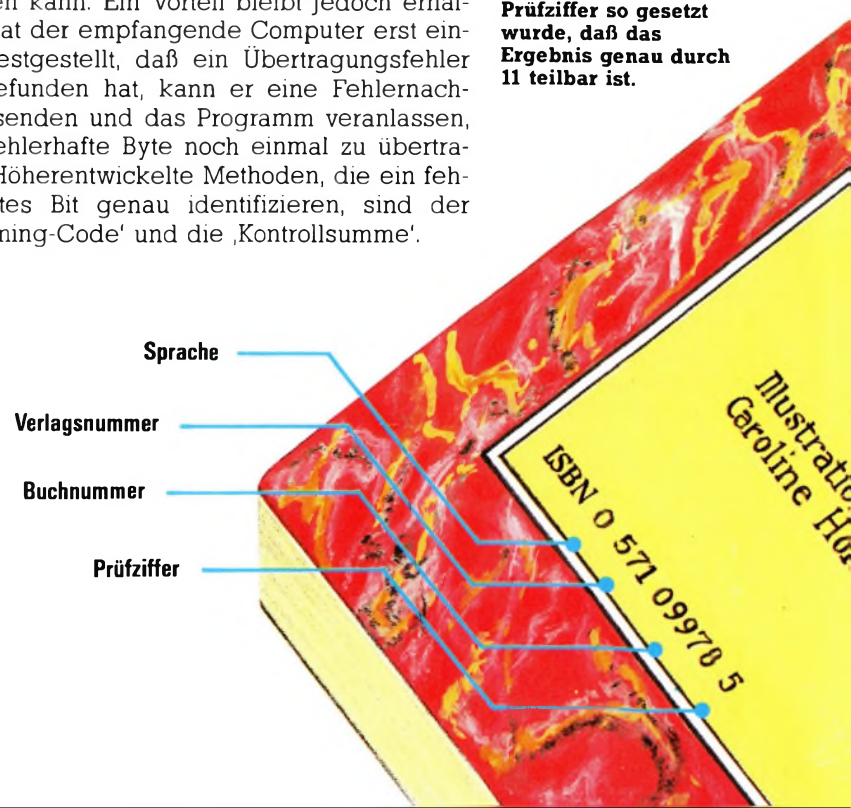
Was geschieht nun bei einem Übertragungsfehler wie im obigen Beispiel? Empfangen wurde:

[1]1000001 (fehlerhafter ASCII-Code für Q).

Wird dieses Byte auf Parität geprüft (entweder durch die Software oder eine spezielle Hardware), stellt sich heraus, daß die Anzahl der 1-Bits ungerade und das Byte demnach fehlerhaft ist. Es kann zurückgewiesen werden. Diese Paritätsprüfung ist selbst dann wirksam, wenn das Prüfbit beim Übertragen kippt.

Betrachten Sie die in Ihrem Computer verwendeten ASCII-Codes. Sie stellen wahrscheinlich fest, daß das achte zwar tatsächlich verwendet wird, aber nicht als Prüfbit. Da Fehler in der internen Datenübertragung sehr selten sind, wird dieses Bit dem Computer für ein zusätzliches Symbol (meist ein Grafiksymbol) zur Verfügung gestellt. Die Paritätsprüfung wird normalerweise nur bei der Datenübertragung oder dann verwendet, wenn Daten auf eine magnetische Oberfläche (Cassette oder Diskette) geschrieben werden, weil diese Vorgänge anfällig für Übertragungsfehler sind.

Die Paritätsprüfung ist eine einfache, aber wirkungsvolle Methode zum Aufspüren von Übertragungsfehlern. Sie findet aber nicht das eigentliche Fehlerbit heraus, und der empfangende Computer ist nicht in der Lage, den Fehler zu korrigieren. Schwerer wiegt jedoch, daß ein Byte, das zwei Fehlerbits enthält, fälschlicherweise als korrekt angenommen werden kann. Ein Vorteil bleibt jedoch erhalten: Hat der empfangende Computer erst einmal festgestellt, daß ein Übertragungsfehler stattgefunden hat, kann er eine Fehlernachricht senden und das Programm veranlassen, das fehlerhafte Byte noch einmal zu übertragen. Höherentwickelte Methoden, die ein fehlerhaftes Bit genau identifizieren, sind der ‚Hamming-Code' und die ‚Kontrollsumme'.



Die letzte Stelle in einer Internationalen Standard-Buch-Nummer (ISBN) ist eine Prüfziffer, vergleichbar mit dem Paritätsbit in einem Computer. Multiplizieren Sie die erste Ziffer der ISBN (0) mit 10, die zweite (5) mit 9 usw. und zählen Sie die Ergebnisse zusammen. Sie werden feststellen, daß die Prüfziffer so gesetzt wurde, daß das Ergebnis genau durch 11 teilbar ist.

Zeichengenerierung

Im Zeichengenerator, einem festgelegten Sektor im Speicher des Computers, wird definiert, wie ein Zeichen auf dem Bildschirm aussieht. Bei manchen Systemen ist es auch möglich, eigene Zeichen zu „konstruieren".

Im BASIC-Kurs haben Sie gelernt, daß jedes alphanumerische und grafische Zeichen als 8-Bit-Code (gewöhnlich im ASCII-Format) fest gespeichert ist. Sollen Zeichen (Buchstaben, Ziffern, Symbole) auf dem Bildschirm erzeugt werden, müssen die Codes für diese Zeichen im Bildwiederholungsspeicher (Video-RAM), einem reservierten Speicherbereich, abgelegt werden. Ein Beispiel: Der Buchstabe A soll in der oberen linken Ecke des Bildschirms erscheinen. In diesem Fall enthält das erste Byte im Video-RAM den Code 65 (ASCII für A). Kommt ein C unter den Buchstaben A, ist der Code 67 (ASCII für C) an 41ster Stelle im Video-RAM abgelegt, vorausgesetzt, der Computer stellt 40 Zeichen pro Zeile dar, und so fort. Wie verwandelt nun der Computer den Code 65 in ein Punktraster, das auf dem Bildschirm den Buchstaben A darstellt? Die Antwort liegt in einer Einrichtung, die ‚Zeichengenerator' genannt wird.

Punktmuster-Sammlung

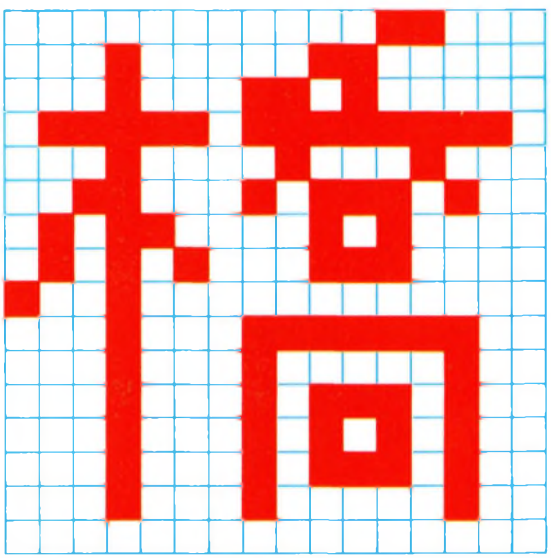
Dieser Generator ist im Prinzip lediglich eine Sammlung von Punktmustern, die als Bits gespeichert sind. Bei Heimcomputern ist der Zeichengenerator im ROM gespeichert. Dabei kann er in den ROMs untergebracht sein, die den BASIC-Interpreter und das Betriebssystem enthalten, oder aber in einem separaten ROM-Chip. Ist der Zeichengenerator in einem separaten Chip untergebracht, besteht die Möglichkeit, auf dem Markt erhältliche Austausch-ROMs einzusetzen, mit denen sich Zeichen in anderer Typographie oder auch spezielle Symbole erzeugen lassen, wie sie beispielsweise von Ingenieuren oder Mathematikern verwendet werden. Bei vielen Computern besteht die Möglichkeit, den Zeichengenerator in den RAM-Speicher zu übertragen.

Jedes Zeichen wird aus einzelnen Punkten zusammengesetzt, deren Position auf einem Raster definiert wird. Die meisten Heimcomputer verwenden ein 8 × 8-Raster. Besser ist jedoch ein größeres Raster, weil durch die damit verbundene höhere Auflösung die Lesbarkeit wesentlich verbessert und der Bereich der darstellbaren Zeichen erweitert wird, wie die Darstellung mit dem 16 × 16-Raster deutlich macht. Die Zeichen entstehen durch Ausfüllen von Feldern in dem das Raster bildenden Gitter. Jedes ausgefüllte Feld wird durch ein 1-Signal, jedes leere Feld durch ein 0-Signal definiert. Für ein Zeichen, das in ein 8 × 8-Punktraster zerlegt ist, wird ein Speicherplatz von acht Bytes (8 × 8 = 64 Bits) benötigt.

Das erste Byte im Zeichengenerator ist der Code für die oberste Rasterzeile des ersten Buchstabens. Bei einem Computer, der nach dem ASCII-Code mit seinen Werten zwischen 0 und 127 arbeitet, benötigt der Zeichengenerator 128 × 8 Bytes.

Wenn nun die Rasterabtastung des Bildschirms die oberste Zeile der Anzeige generiert, muß der Computer die oberste Zeile der Punktmatrix für den Buchstaben erzeugen, der oben links auf dem Bildschirm steht, dann die oberste Punktzeile des rechts daneben stehenden Buchstabens und so fort bis an das Bildschirmende.

Kehren wir zu unserem Beispiel mit dem Buchstaben A zurück, dessen ASCII-Code 65 ist. Wir können ausrechnen, daß die erste Rasterzeile in Byte Nummer 520 (65 × 8 + 0) gespeichert ist, die zweite in Byte Nummer 521 (65 × 8 + 2) und so weiter. Der Video-Schaltung bleibt jetzt nur noch, diese acht Bits in eine Spannungsfolge umzuwandeln, die den abtastenden Elektronenstrahl ein- und ausschaltet, um die Anzeige des Zeichens auf dem Bildschirm zu erzeugen.



Japanische Schriftzeichen sind sehr kompliziert. Eine gewöhnliche 8 × 8-Matrix kann nicht genügend Einzelheiten darstellen, um diese Schriftzeichen lesbar zu machen. Das hier gezeigte japanische Schriftzeichen für „Brücke" ist erst auf einer 16 × 16-Matrix lesbar. Noch deutlicher werden Schriftzeichen mit einer 24 × 24-Punktmatrix.

橋

Schritt für Schritt

Beim Messen linearer oder winkelförmiger Bewegungen mit Hilfe optischer Sensoren erweisen sich Binärcodes als fehlbar. Darum wurde der Gray-Code entwickelt.

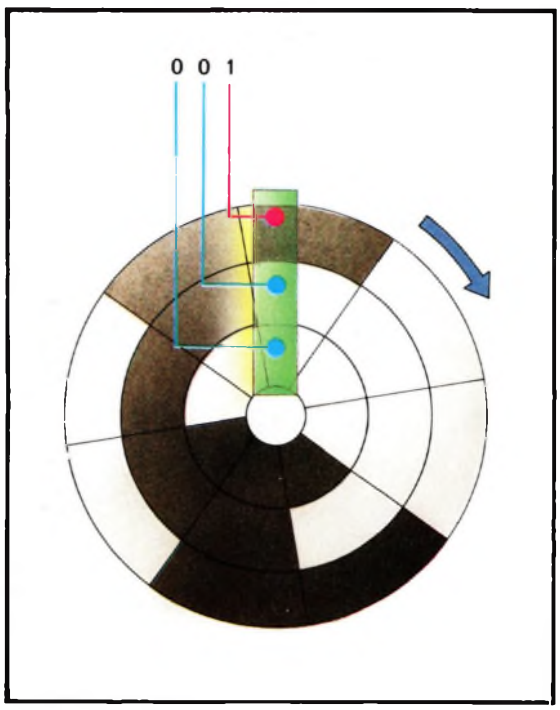
Nur ein Bit ändert sich
Die Tabelle zeigt den Gray-Code im Vergleich zum Binärcode für Dezimalzahlen von 0 bis 15.

Dezimal	Binär	Gray-Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Bei zahlreichen Anwendungen ist es notwendig, die genaue örtliche Position eines beweglichen Gegenstandes festzustellen und die Werte an einen Computer zu übermitteln. Beispiele dafür sind Roboter, bei denen der Computer die Positionen und Bewegungsrichtungen aller Elemente kennen muß, oder Werkzeugmaschinen, wo die Frästischbewegungen von einem Computer gesteuert werden. Wie aber wird eine Position in eine Binärzahl verwandelt, die vom Computer verarbeitet werden kann?

Eine Lösung wäre die Verwendung eines Analogsystems. Das sich bewegende Element wird an einen Regelwiderstand angeschlossen und die resultierende Spannung in einen Analog-Digital-Umsetzer eingelesen (oder direkt in den Analogeingang des Computers). Ein solches System ist jedoch nicht sehr genau und seine mechanischen Elemente unterliegen dem Verschleiß.

Ein anderer Weg ist, einen Binärcode auf das entsprechende Element zu drucken und diesen direkt in den Computer einzulesen. Der Code besteht gewöhnlich aus einer Reihe schwarzer und weißer Felder und wird mit Hilfe eines auf die Felder gerichteten Lichtstrahls in Verbindung mit Fotozellen gelesen.



Die Winkelstellung eines Gegenstandes kann in den Computer mit Hilfe einer Scheibe eingelesen werden, auf die ein Code gedruckt ist. Das durch einen Lichtstrahl angeleuchtete Rasterbild wird durch eine Reihe Fotozellen abgetastet. Bewegt sich der Gegenstand, verändert sich das Rasterbild unter den Fotozellen, und es wird ein digitales Signal erzeugt. Bei Verwendung eines Binärcodes kann es zu fehlerhaften Ausgangssignalen kommen, wenn die Scheibe am Übergang von einem Rasterbild zum andern stehen bleibt. Diese Schwierigkeit wird durch den Gray-Code überwunden.

Je eine Fotozelle ist dabei für eine Stelle des Binärmusters verantwortlich. Bewegt sich das Teil, verändert sich das Binärmuster unter den Fotozellen und erzeugt dadurch ein Binär-Signal, das die gegenwärtige Position des Elementes bestimmt. Neben linearen Binärmustern gibt es auch kreisförmige zur Erfassung von Drehbewegungen, wie z. B. in einem Roboter-Armgelenk.

Korrekte Signale

Schwierigkeiten entstehen hierbei, wenn sich der Gegenstand von dem einen Binärcode zum anderen bewegt oder wenn er gar zwischen zwei Codes zum Stehen kommt, denn dann können die Fotozellen sowohl den einen, als auch den anderen Code lesen. Bleibt das Element beispielsweise zwischen der Binärposition 11 (1011) und 12 (1100) stehen, kann nur das höchstwertige Bit (die am weitesten links stehende 1) ein korrektes Signal liefern, wogegen die anderen drei Bits unzuverlässige Werte wiedergeben. In bestimmten Situationen ändern sich alle Bits, wie beim Übergang von binär 7 (0111) auf binär 8 (1000), so daß selbst kleine Ungenauigkeiten im Muster zu unkorrekten Werten aller drei Fotozellen führen können. Die Positionsangabe wäre möglicherweise völlig falsch, und der Computer würde dies nicht erkennen. Die Folgen könnten verhängnisvoll sein.

Um diese Schwierigkeiten zu beheben, wird ein Zählsystem benötigt, in dem sich bei jedem Schritt nur ein Bit ändert. Dies stellt sicher, daß beim Übergang von einem Code zum andern nur ein Bit fehlerhaft sein und sich ein Positionsfehler nur im Rahmen eines Schritts bewegen kann. Ein solches Zählsystem ist der sogenannte Gray-Code, der unmißverständlich durch folgende Anforderungen bestimmt wird: Bei jedem Bewegungsschritt ändert sich nur ein einziges Bit, und dies soll das am weitesten rechts stehende Bit sein, das noch ein unverwechselbares Muster ergibt. Beginnen wir, wie im Binärsystem, mit 0000, wird die 1 im Gray-Code durch 0001 repräsentiert. Um die 2 darzustellen, muß nun das zweite Bit geändert werden, um 0011 zu erhalten. Zur Darstellung der 3 ist es jetzt möglich, das erste Bit zu ändern, um 0010 zu erhalten. Demgegenüber wäre die Binärfolge für diese Zahlen 0000, 0001, 0010 und 0011.

Speicherstruktur

Höhere Programmiersprachen wie beispielsweise BASIC verwalten den Speicher automatisch. Detaillierte Kenntnisse über den Speicheraufbau helfen, die Fähigkeiten des Computers optimal auszunutzen.

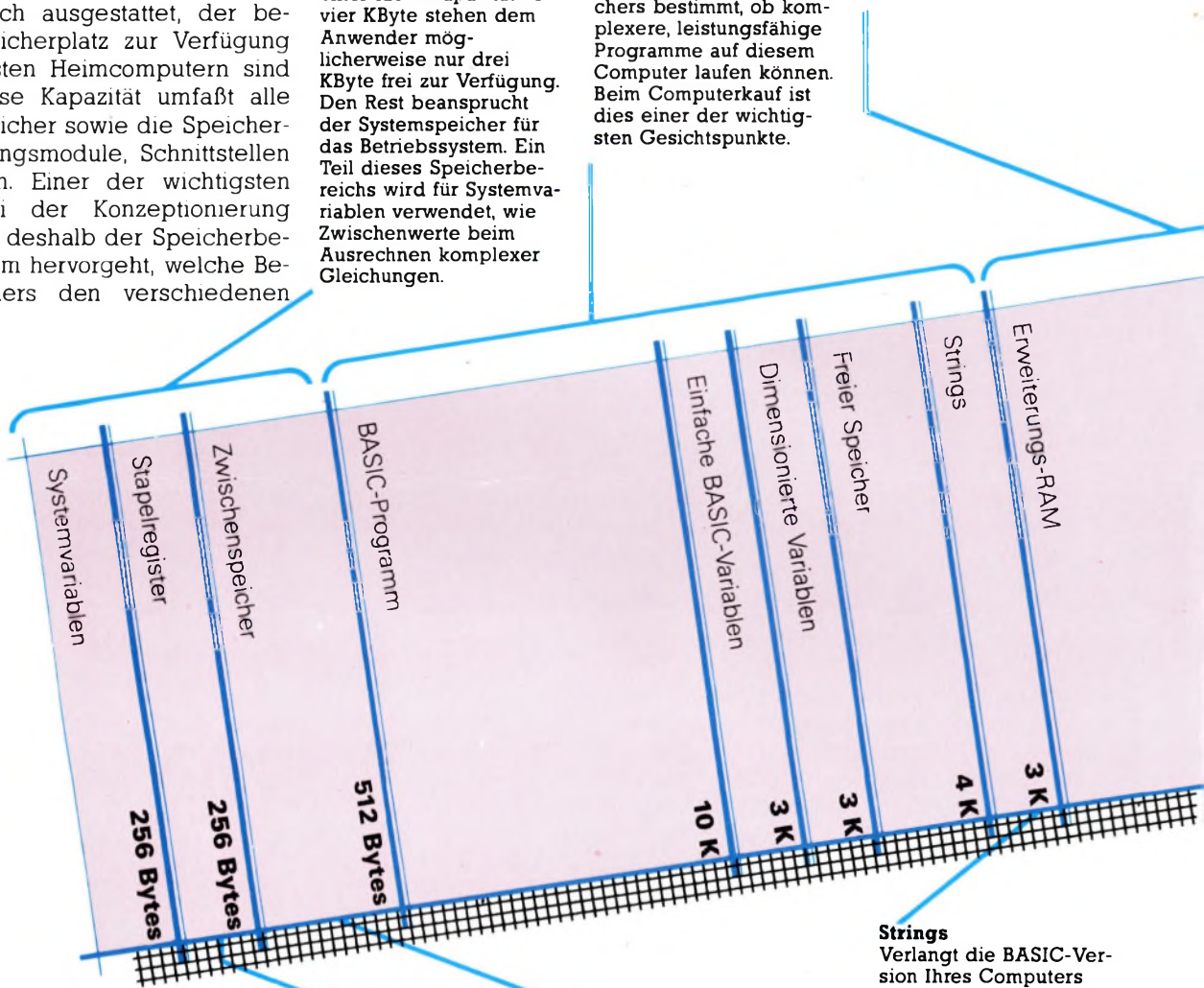
Die Zentraleinheit CPU (Central Processing Unit) eines Computers ist mit einem festen Adressenbereich ausgestattet, der bestimmt, wieviel Speicherplatz zur Verfügung steht. Bei den meisten Heimcomputern sind dies 64 KByte. Diese Kapazität umfaßt alle RAM- und ROM-Speicher sowie die Speicherplätze für Erweiterungsmodule, Schnittstellen und Anschlußstellen. Einer der wichtigsten Gesichtspunkte bei der Konzeptionierung eines Computers ist deshalb der Speicherbelegungsplan, aus dem hervorgeht, welche Bereiche des Speichers den verschiedenen Rechnerfunktionen zugeordnet sind. Wer sich mit BASIC-Programmierung begnügt, braucht sich um die Speicherorganisation nicht zu kümmern, wer aber in die Maschinensprache einsteigen oder Hardware bauen möchte, muß die Speicherorganisation genau kennen.

Hier wird gezeigt, wie ein typischer Speicherbelegungsplan aussieht. Dabei stützen wir uns mehr auf einen Computer mit einem 6502-Prozessor als CPU als auf einen Z80-Prozessor. Leider liefern nicht alle Computerhersteller genaue Angaben über die Speicherbelegung. Gewöhnlich läßt sich jedoch die Speicherorganisation durch Experimentieren herausfinden.

Systemspeicher
Bei einem Computer mit einer RAM-Kapazität von vier KByte stehen dem Anwender möglicherweise nur drei KByte frei zur Verfügung. Den Rest beansprucht der Systemspeicher für das Betriebssystem. Ein Teil dieses Speicherbereichs wird für Systemvariablen verwendet, wie Zwischenwerte beim Ausrechnen komplexer Gleichungen.

Anwender-RAM
Der Umfang dieses Speichers bestimmt, ob komplexere, leistungsfähige Programme auf diesem Computer laufen können. Beim Computerkauf ist dies einer der wichtigsten Gesichtspunkte.

Nicht belegt
Im Speicher muß auch Platz für Erweiterungs-RAMs reserviert sein. Es gibt Computer, die eine Erweiterung von mehr als 64 KByte erlauben, wobei eine besondere Schaltung die entsprechende RAM-Sektion je nach Bedarf in diesen Speicherbereich schaltet.



Stapelregister
Dieser Bereich ist ausschließlich für die CPU reserviert und als „LIFO" (Last In/First Out) strukturiert. Wird beispielsweise eine GOSUB-Routine in BASIC ausgeführt, schiebt die CPU denjenigen Speicherplatz im Register nach oben, auf den sie zurückkehren (RETURN) muß. Das Stapelregister wird besonders stark bei arithmetischen Rechnungen und in FOR ... NEXT-Schleifen in Anspruch genommen.

Zwischenspeicher
Für die Tastatur muß ein Zwischenspeicher reserviert sein, der Zeichen aufnimmt, die schneller eingegeben werden, als sie das Programm verarbeiten kann. Auch Peripheriegeräte, wie z. B. Cassettenrecorder, benötigen einen Zwischenspeicher, weil die meisten Betriebssysteme die Daten blockweise übertragen.

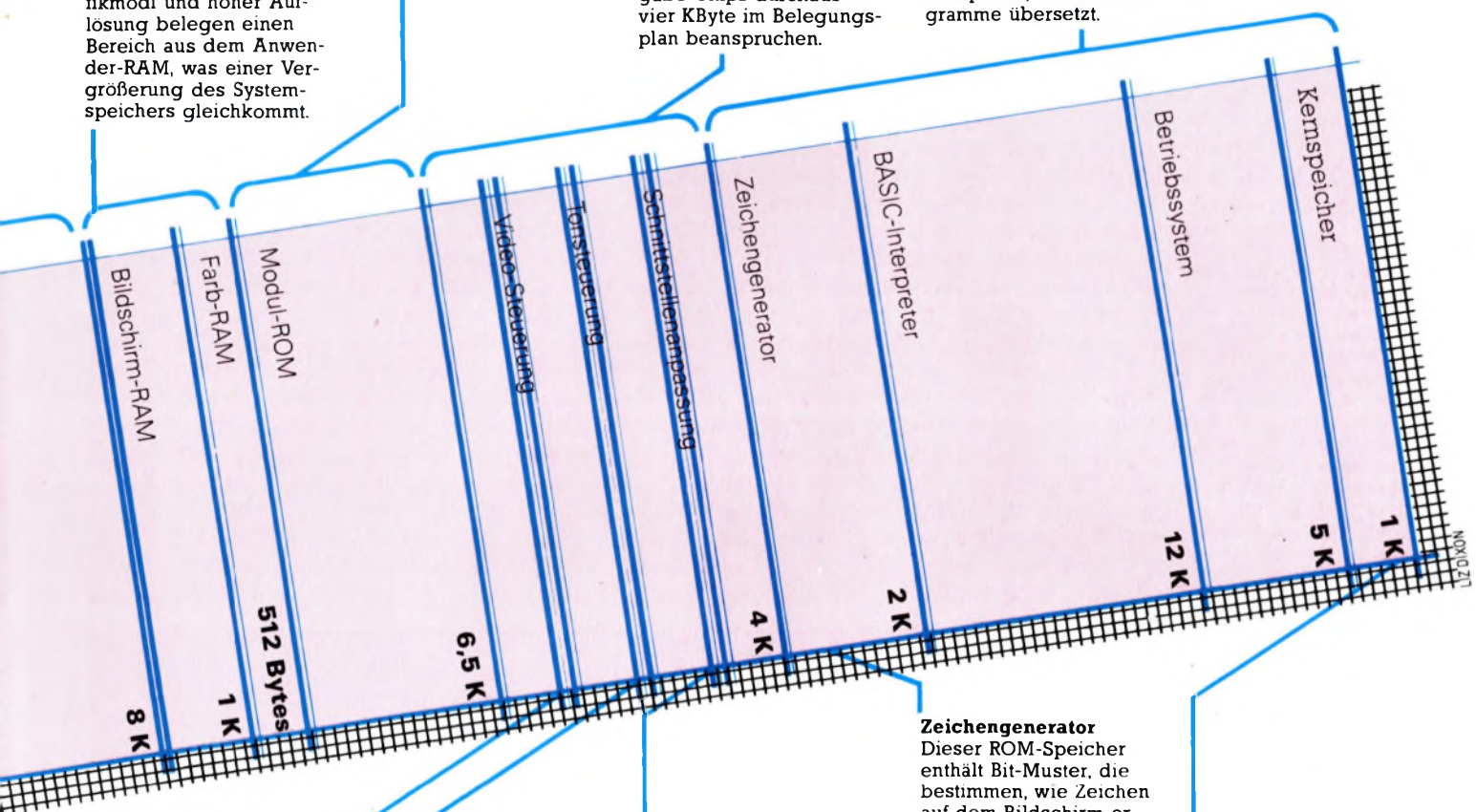
Strings
Verlangt die BASIC-Version Ihres Computers eine Dimensionierung der benötigten Strings, werden diese nach dem gleichen Schema abgelegt wie dimensionierte Variablen. Sind jedoch „dynamische" Strings möglich, deren Länge veränderbar ist, werden die tatsächlichen Daten getrennt in einem Speicherteil abgelegt, dessen Größe variabel ist. In gewissen Zeitabständen sorgt dann das Betriebssystem für eine „Müllaktion", d. h. nicht mehr benötigte Daten werden gelöscht.

System-RAM
Manche Systeme weisen diesen RAM-Speicher nicht als Teil des Anwender-RAM-Speichers aus. Dies trifft gewöhnlich auf das Bildschirm-RAM zu, in dem ein Byte je einem Zeichenplatz auf dem Bildschirm entspricht, und ebenso für das Farb-RAM, in dem ein Byte die Vorder- und Hintergrundfarben für die Zeichenpositionen bestimmt. Computer mit vielen Grafikmodi und hoher Auflösung belegen einen Bereich aus dem Anwender-RAM, was einer Vergrößerung des Systemspeichers gleichkommt.

Nicht belegt
Programme auf Steckmodulen (Cartridges) werden im Speicherbelegungsplan als Erweiterungs-ROM behandelt. Manche Computer haben eigene ROM-Steckplätze für zusätzliche Sprachen, für die der Belegungsplan ebenfalls einen reservierten Bereich ausweist.

Eingabe-/Ausgabe-Chips
Eine CPU kann nur mit Einrichtungen kommunizieren, die als Speicherplatz im Belegungsplan erscheinen. Darin müssen alle Schnittstellen und Chips im Belegungsplan enthalten sein. Dazu gehören die Schnittstellen für die Tastatur, das Cassettenlaufwerk, die Videosteuerung und den Drucker. Da die CPU Speicherplätze blockweise adressiert, können die Ein-/Ausgabe-Chips durchaus vier KByte im Belegungsplan beanspruchen.

System-ROM
Der ROM-Speicher enthält Informationen, die permanent benutzt, jedoch nicht verändert werden können. Der wichtigste Teil des ROM-Speichers ist das Betriebssystem, das die Funktionen des Computers steuert. Diese Maschinenprogramme führen Funktionen wie Abfragen der Tastatur und Speichern oder Lesen von Daten auf Cassette aus. Eine andere Komponente ist der BASIC-Interpreter, der die Programme übersetzt.



Video-Steuerung
Hochentwickelte Grafiken, wie Sprites und Mehrfachauflösungen, werden zunehmend über die Hardware und nicht über die Software gesteuert. Im Belegungsplan erscheint die Video-Steuerung als eine Anzahl von Einbyte-Registern, die jede sichtbare Komponente bestimmen, angefangen von der Hintergrundfarbe bis zur genauen Position eines jeden Sprites.

Tonsteuerung
Einfache Toneffekte können über die Software erzielt werden. Computer mit großer Tonpalette oder eigenem Tongenerator verfügen jedoch über eine besondere Tonsteuerung mit eigenem Verstärker.

Schnittstellenanpassung
Die Schnittstellenanpassung PIA (Peripheral Interface Adaptor) dient dazu, die üblichen Schnittstellen für Tastatur, Cassetten, Joysticks (Steuerknüppel) und Drucker zu handhaben. Die meisten hochentwickelten Chips können zwischen parallelen und seriellen Daten unterscheiden und haben eingebaute Zeitschalter, die beim Programmieren oder dazu, Übertragungsraten zu steuern, verwendet werden können.

Zeichengenerator
Dieser ROM-Speicher enthält Bit-Muster, die bestimmen, wie Zeichen auf dem Bildschirm erscheinen. Bei manchen Computern können alle oder ein Teil der gesetzten Zeichen in den RAM-Speicher kopiert werden, was für den Anwender die Möglichkeit schafft, selbst Zeichen zu definieren.

Kernspeicher
Dieser Speicher, der bei nahezu jedem Computer einen anderen Namen trägt, ist das Herz des Betriebssystems. Beim Einschalten des Computers springt die CPU automatisch zu dieser Speicherstelle und führt das „Quellprogramm" aus. Die CPU sucht den RAM-Bereich nach der verfügbaren Speicherkapazität ab und prüft, ob ein Cartridge eingesteckt ist. Vom Kernspeicher werden auch die meisten elementaren Formen der Eingabe und Ausgabe gehandhabt.

Optimierungen

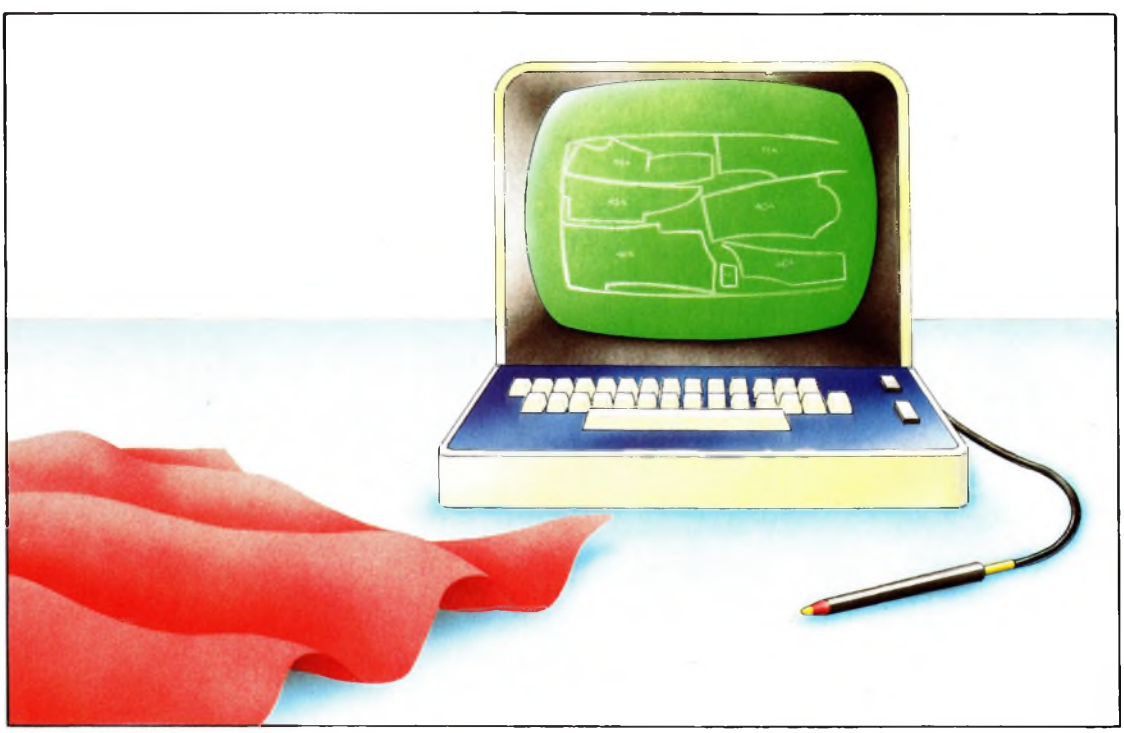
Die Suche nach der optimalen Lösung erfordert häufig komplexe Mathematik. Mit fortschreitender Technik übernehmen Computer diese Arbeit.

Bei jeder Entscheidung, die Sie treffen, ist unweigerlich ein Kompromiß enthalten – zum Beispiel zwischen Kosten und Leistung oder zwischen Kosten und Zeit. Es ist unwahrscheinlich, daß die absolut beste Lösung zu den absolut geringsten Kosten erreicht werden kann. Das „optimale" Ergebnis wird irgendwo in der Mitte zwischen beiden Möglichkeiten liegen.

Folgende Überlegungen würden Sie zum Beispiel bei der Auswahl zwischen zwei verschiedenen Waschmitteln anstellen: „650 Gramm dieses Pulvers kosten 2,16 DM, 1300 Gramm des anderen kosten 3,89 DM. Wenn ich nun aber 20 Prozent mehr vom billigeren Waschmittel benötige, um das gleiche Ergebnis zu erhalten, welches Mittel ist dann preiswerter?" Falls Sie diese Gedanken in Form einer einfachen Gleichung – in diesem Fall durch Prozentdifferenzen – ausdrücken, ist die Antwort leicht zu ermitteln.

Sie könnten aber auch den verschiedenen Faktoren unterschiedliche Bedeutung beimessen. (Vielleicht ist Ihnen die Kostenersparnis wichtiger als das Waschergebnis.) Die Lösung müßte dann natürlich anders aussehen. Das Prinzip, eine Berechnung mit einem konstanten Wert zu „gewichten", funktioniert gut, wenn die Differenzen zwischen gleichen Größen (zum Beispiel Preis und Gewicht) ebenfalls konstant sind. Wenn aber diese Differenzen sich in unterschiedlichem Maß verändern, wird logischerweise auch die erforderliche Mathematik komplizierter. Sind nur wenige Faktoren zu berücksichtigen, dann eignet sich die Methode der Matrizenrechnung. Bei größeren Mengen von veränderlichen Daten wird die lineare Optimierung angewendet. Ein anderer Weg ist, die Lösung abzuschätzen und dann den Wert schrittweise zu verändern, bis alle Bedingungen erfüllt sind (Näherungsverfahren). Je besser die erste Schätzung ist, um so schneller wird dieser Prozeß zum Ziel führen.

Optimierungsstrategien wie die genannten sind für Handel und Industrie außerordentlich wichtig. Sie werden besonders in den Bereichen Serienfertigung und Baugewerbe eingesetzt. Lineare Programmierung, CPA (Critical Path Analysis) und PERT (Program Evaluation Research Technique) sind einige der Namen für solche Methoden. In ihren ursprünglichen Formen tauchten sie etwa 30 Jahre vor Anbruch des Computerzeitalters auf. Die Methoden waren sehr arbeitsaufwendig. Heute können Mi-



Das Anordnen von Schnittmusterteilen auf einer Stoffbahn, so daß möglichst wenig Abfall entsteht, ist ein gutes Beispiel für rechnerunterstützte Optimierung. Hier zeigt der Computer seinen Vorschlag auf dem Schirm, und ein erfahrener Bediener kann Korrekturen mit Hilfe eines Lightpens durchführen.

	Preise in Pfennigen	Eiweiß g	Kohlenhydrate g	Fett g	Kilojoule	Benötigte Menge
400 g Kartoffeln	30	8	—	64	1204	10 kg
200 g Sardinen	150	41	49	1	2554	—
400 g Steak	720	84	28	0	4856	—
1/2 Liter Milch	60	16	18	24	945	6 l
Mindestbedarf pro Woche		360	215	505	36925	

Die optimale Diät

Hier soll die beste Kombination von vier Nahrungsmitteln zu einer vorgegebenen Minimaldiät bei möglichst geringen Kosten gefunden werden. Hierzu geben Sie dem Computer ein: die Nährwerte (rosa) und den Preis jedes Nahrungsmittels (blau) sowie den Minimalbedarf an jedem Nährstoff pro Woche (gelb). Der Rechner bestimmt das kritischste Element und manipuliert den Rest des Rasters entsprechend, um die optimale Verteilung (grün) zu finden. In diesem Beispiel wird der Bedarf ausschließlich mit Milch und Kartoffeln gedeckt, bei knappen Kosten von 14,70 DM.

crocomputer solche Aufgaben übernehmen.

Eine Branche, die beträchtlich von der Entwicklung der Optimierungsmethoden profitiert hat, ist die Textilindustrie. Stoffe werden üblicherweise in standardisierten Breiten geliefert. Und das Problem des Herstellers liegt darin, den Anfall von Verschnitt beim Zuschneiden des Stoffes zu minimieren und gleichzeitig wichtige Faktoren, wie die Laufrichtung des Gewebes oder die Musterung, zu beachten.

Verbesserte Muster

In den modernen Textilfabriken Europas wird die Plazierung von Schnittmustern für Konfektionskleidung auf einem vorgegebenen Stück Stoff mit Optimierungstechniken berechnet, und das vorgeschlagene Ergebnis wird auf einem Bildschirm angezeigt. An diesem Punkt wird der Computerbediener benötigt, um sein Urteil und seine Erfahrung zur Verbesserung der Berechnung des Computers anzubringen. Der Anwender macht durchschnittlich bei einem von fünf Mustern eine Verbesserung.

Weil die Anforderungen bei jedem Auftrag und bei jedem Stoff anders sind, ist dies eine sinnvolle Anwendung der mit der Erfahrung des Bedieners kombinierten Computeroptimierung. Noch weitergehende Methoden finden in Werken Anwendung, die wiederholt identische Objekte aus dünnem Material schneiden. Da der Vorgang des Schneidens oder Stanzens ein Teil der Fließbandproduktion ist, wird ein und derselbe Arbeitsgang tausendfach wiederholt. In diesem Fall werden die Kosten des Optimierens durch die Einsparungen an Material bei weitem aufgewogen.

Die CPA ist eine Methode, um die wichtigste Reihenfolge von Tätigkeiten in einem Herstellungs- oder Bauvorhaben herauszufinden. So werden die Arbeiten bestimmt, deren nicht termingerechter Abschluß am ehesten dazu führt, daß alles andere aufgehalten wird. Die Methode ist stark zeitorientiert. Der Wert eines Vorgangs im CPA-Diagramm wird durch die Zeit, die zu seiner Durchführung benötigt wird, bestimmt. Üblicherweise wird die Methode im Planungsstadium von Bauprojekten angewendet, damit die Planer Menschen und Materialien für die verschiedenen Abschnitte des Projektes in der richtigen Reihenfolge einsetzen können – Installationsarbeiten bevor der Fußboden verlegt wird, Verputzen vor dem Streichen. Auch hierfür ist Software für viele Microcomputertypen verfügbar.

Die Art und die Streckenführung von Straßen, ob in der Stadt oder auf dem Land, ist weitgehend von Optimierungstechniken abhängig. Der Architekt wird sich vor allem um Steigungen und Kurvenradien kümmern. Der Landwirt, dessen Boden benötigt wird, gibt völlig andere Punkte zu bedenken. Wenn eine neue Straße geplant wird, muß eine große Datenmenge gesammelt werden, um ein umfassendes Modell der Situation zu erstellen. Anhand des Modells wird die Streckenführung optimiert.

Variabler Chip

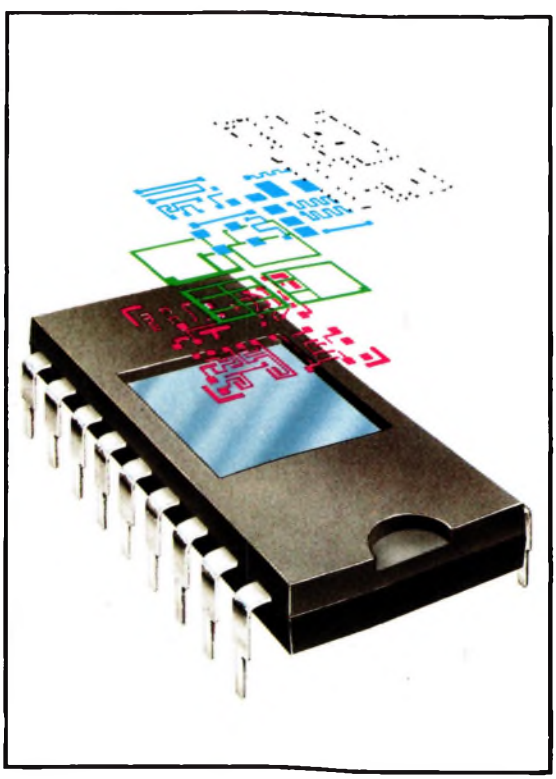
„Ungebundene" Logikschaltungen, kurz ULA genannt, steuern als Schaltzentrale zahlreiche Funktionen des Computers.

Die Microcomputer-Schwemme hat zahlreiche Fortschritte auf dem Gebiet der Elektronik hervorgebracht. Eine der wichtigsten Entwicklungen ist der ULA (Uncommitted Logic Array), ein Chip, der nicht an einen bestimmten Zweck gebunden ist. Diese Entwicklung hat inzwischen einen Stand erreicht, der es möglich macht, hochentwickelte Computer und andere Einrichtungen mit nicht mehr als vier Hauptbestandteilen zu bauen: CPU, RAM, ROM und ULA, der diese Komponenten miteinander verbindet.

Ein ULA besteht aus einer Vielzahl von Logikgattern, die ursprünglich nicht an einen bestimmten Zweck gebunden sind, sich aber so miteinander verbinden lassen, daß sie nahezu jede gewünschte Operation durchführen können. Der ULA ist eine Art ROM; sein Inhalt kann, wie beim ROM, nur vom Hersteller, nicht aber vom Anwender festgelegt werden.

Vor dem Programmieren ist ein ULA nichts weiter als eine Anhäufung einfachster elektronischer Schaltungen, die nicht miteinander verknüpft sind. Sie können darum auch keine Tätigkeiten ausüben. Der ULA besteht aus Halbleiterschichten mit einer Deckschicht aus leitendem Material, die die Verbindungen zwischen den Einzelschaltungen herstellt. Obwohl jede Einzelschaltung ganz einfach ist und vielleicht nur aus einigen Transistoren oder einem einzigen Widerstand bestehen mag, ergibt sich eine große Zahl möglicher Verbindungen, die dem ULA eine außergewöhnliche Flexibilität verleihen und selbst so komplexe Schaltkreise wie ein Flip-Flop entstehen lassen. Solche Schaltkreise, auch „Module" genannt, lassen sich meist aus weniger als einem halben Dutzend Einzelschaltungen bilden.

Ein ULA kann auf ein außergewöhnlich breittes Spektrum von Funktionen programmiert werden. Er kann als Tongenerator arbeiten, Verschlußzeit, Schärfe und Motor in einer Kamera steuern oder den größten Anteil der Arbeit in einem Digitalthermometer leisten. Dabei benötigt der ULA keine Außenverdrahtung – ausgenommen die Anschlüsse für Batterien, Schalter oder Sensoren.



Jeder Halbleiter-Chip besteht aus Schichten, die individuell so geätzt sind, daß Schaltelemente entstehen. Die oberste Schicht bestimmt die Verbindung zwischen den einzelnen Schaltelementen. Ein ULA besteht aus vielen Logikelementen, die so kombiniert werden können, daß komplexe Schaltungen entstehen.

Große Möglichkeiten

Bei der Konstruktion der Deckschicht, die die Einzelschaltungen verbindet, werden Rechner eingesetzt. Der Mini-Computer eines CAD-Systems, z. B. der DEC PCP11/23, zeichnet zunächst ein Diagramm der gewünschten Logik auf und entwickelt dann daraus das „Layout", die gedachte Anordnung der Einzelschaltungen. Der Konstrukteur führt dabei das System von einem Grafik-Terminal aus und kann sich das Ergebnis vom Plotter ausdrucken lassen.

Ist der Schaltplan fertig, wird er auf einen größeren Computer übertragen. Dieser prüft ihn auf Übereinstimmung mit der ursprünglichen Logik, forscht nach gravierenden Fehlern und stellt auf diese Weise fest, ob der Schaltplan akzeptabel ist. Als nächstes folgt die Simulation der Schaltung mit Hilfe eines vom Anwender zur Verfügung gestellten Testprogramms. Verläuft die Simulation zufriedenstellend, kann der Computer die Vorlagen für die zum Herstellen der Deckschicht erforderlichen Masken produzieren.

Wie weit gehen die Möglichkeiten eines ULA? Der Gedanke, eine Reihe einfacher Schaltungen auf Silizium festzulegen und es dem Anwender zu überlassen, seine Schaltverbindungen selbst herzustellen, ist verlokkend. Dies könnte die bevorzugte Methode zum Herstellen von Schaltkreisen werden. Beim derzeitigen Stand der Technik sind ULAs jedoch nur dann wirtschaftlich, wenn einige tausend identischer Schaltkreise in einem Gerät benötigt werden.

Shell-Sortierung

Die Shell-Sortierung arbeitet nach dem Prinzip der Kettenbildung. Bei großen Datenmengen eignet sich dieses Verfahren besser als die Bubble- oder Vergleichssortierung.

Das nach seinem Erfinder D. Shell benannte Verfahren hat den Vorzug, daß die ungeordneten Informationen in einem Datenfeld frühzeitig geordnet werden und die einzelnen Daten nicht weit von ihren wirklichen Positionen entfernt sind. Außerdem bietet dieses Verfahren die Möglichkeit, Daten über relativ lange Strecken auszutauschen. Die Anwendung ist bei großen Datenmengen sinnvoll, die nicht mit einem Blick überschaut werden können. Um zu sehen, wie die Methode arbeitet, genügt jedoch das folgende einfache Beispiel „Sortieren von Spielkarten":

1) Legen Sie alle Karten einer Farbe in gemischter Reihenfolge aus. Diese Karten sollen jetzt in fallender Ordnung (König ganz links, As ganz rechts) ausgelegt werden. Gehen Sie folgendermaßen vor: Zählen Sie die Karten (hier 13), und teilen Sie die Summe durch zwei. Das Ergebnis, in diesem Fall sechs, ist die „Kettenzahl".

2) Legen Sie über die äußerste linke Karte (Position Nr. 1 innerhalb der Kartenreihe) und über die Karte der Position Nr. 6 (dies entspricht der Kettenzahl) ein Markierungszeichen, zum Beispiel eine Münze. Jede Karte der Positionen eins bis sechs stellt nun eine „linke Eckkarte" der jetzt zu bildenden Ketten dar.

Karten positionieren

Bilden Sie nun die erste Kette: Beginnen Sie mit der Karte der Position eins und addieren Sie zur eins die Kettenzahl sechs, um die Position der nächsten Karte zu erhalten (Nr. 7). Addieren Sie nun zur sieben wieder sechs. Sie erhalten die Karte Nr. 13.

3) Sortieren Sie die Karten der ersten Kette nach ihrer Wertigkeit, und legen Sie diese auf die freien Plätze zurück.

4) Bilden Sie jetzt die zweite Kette auf die gleiche Weise: Position Nr. 2+6=8. Dann wieder sortieren und zurücklegen. Wiederholen Sie diesen Vorgang, bis sechs Ketten gebildet sind. In der ersten Kette finden sich die Karten Nr. 1, 7 und 13, in der zweiten Kette die Karten Nr. 2 und 8 und so weiter.

5) Teilen Sie die Kettenzahl nun durch 2 (6/2=3), und kennzeichnen Sie die Positionen eins, drei, sechs, neun und zwölf. Bilden Sie Ketten wie zuvor, diesmal jedoch mit der Kettenzahl 3.

6) Teilen Sie die Kettenzahl 3 nochmals durch zwei. Sie erhalten unter Vernachlässigung des Restes die Kettenzahl 1. Die jetzt vorliegende Kartenreihe ist folglich die letzte Kette. Sortieren Sie diese Kette nach dem Vergleichsverfahren. Die Kartenreihe muß jetzt geordnet sein, denn ein weiteres Teilen der Kettenzahl durch zwei würde eine Zahl kleiner als eins als Zeichen dafür ergeben, daß ein weiteres Ordnen nicht mehr nötig ist.

```
7999 REM**********************
8000 REM*        SHELL       *
8001 REM**********************
8025 PRINT "SHELL SORT - GO !!!!!"
8050 LET LK=LT
8100 FOR Z=0 TO 1 STEP 0
8150 LET LK=INT(LK/II)
8200 FOR LB=1 TO LK
8250 LET LL=LB+LK
8300 FOR P=LL TO LT STEP LK
8350 LET D=R(P)
8400 FOR Q=P TO LL STEP-LK
8450 LET R(Q)=R(Q-LK)
8500 IF D<=R(Q) THEN LET R(Q)=D:LET Q=LL
8550 NEXT Q
8600 IF D>R(LB) THEN LET R(LB)=D
8650 NEXT P
8700 NEXT LB
8750 IF LK=1 THEN LET Z=1
8800 NEXT Z
8850 PRINT "SHELL SORT - STOP !!!!!"
8900 RETURN
```

Um diese Routine in das Sortierprogramm auf Seite 159 einzufügen, sind die Zeilen 350 und 900 folgendermaßen zu ändern:

```
350 LET I=1:LET O=0:LET II=+:LET TH=3

900 ON SR GOSUB 6000,7000,8000
```

Shell-Sortiertafel

Position Nr.									Kettenzahl	Vorgang
1	2	3	4	5	6	7	8	9		
2	8	9	3	10	5	K	6	7	(9/2)=>4	Anfang Durchgang
*	+	@	$	*	+	@	$	*		Ketten bilden
10				7				2		Kette 1 sortieren
	8				5					Kette 2 sortieren
		K				9				Kette 3 sortieren
			6				3			Kette 4 sortieren
10	8	K	6	7	5	9	3	2		Anfang Durchgang
10	8	K	6	7	5	9	3	2	(4/2)=>2	Ende Durchgang
*	+	*	+	*	+	*	+	*		Ketten bilden
K		10		9		7		2		Kette 1 sortieren
	8		6		5		3			Kette 2 sortieren
K	8	10	6	9	5	7	3	2		Ende Durchgang
K	8	10	6	9	5	7	3	2	(2/2)=>1	Anfang Durchgang
*	*	*	*	*	*	*	*	*		Kette 1 bilden
K		10	9	8	7	6	5	3	2	Ende Durchgang

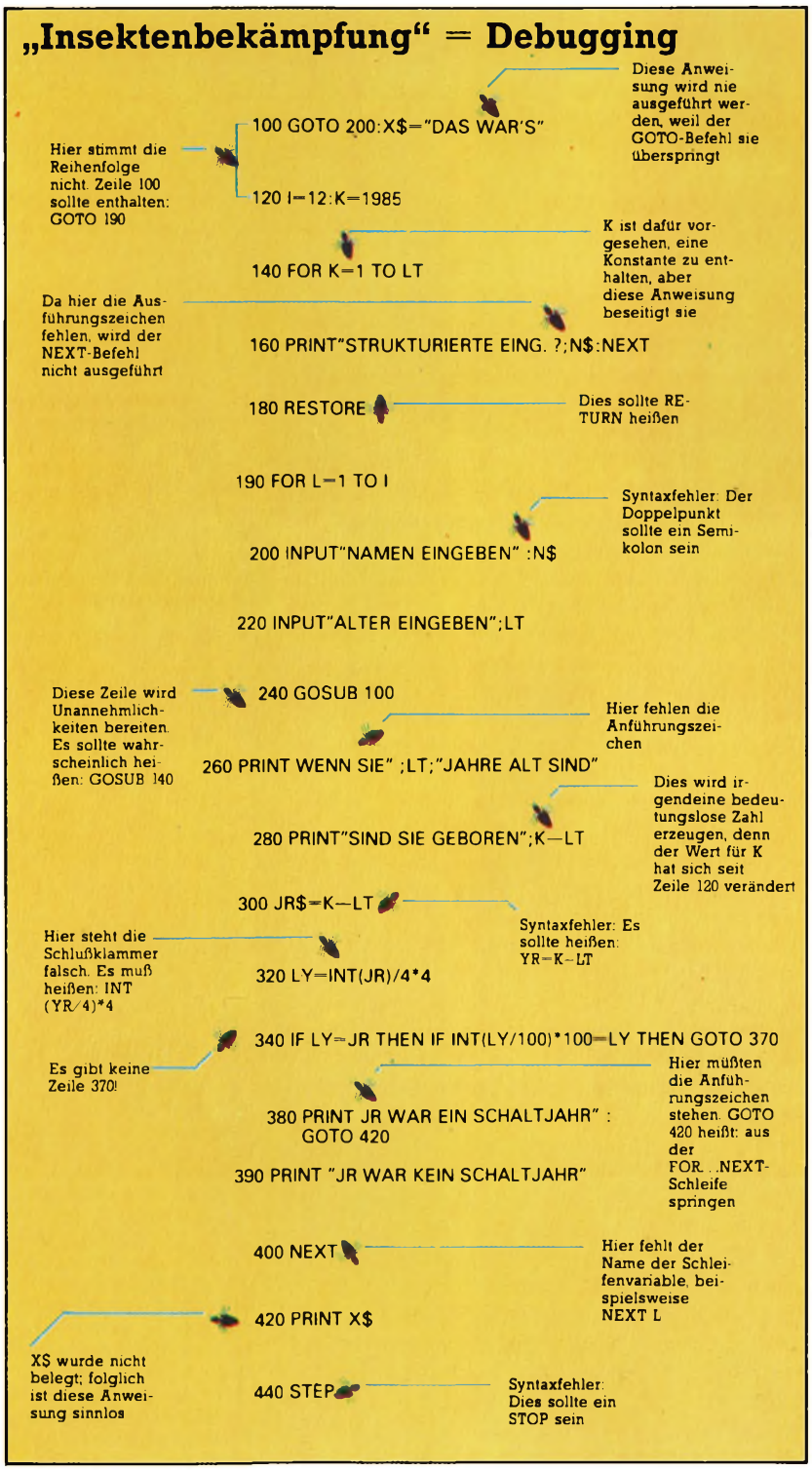
Das links gezeigte Listing für eine Shell-Sortierung ist in Verbindung mit dem Programm auf Seite 159 zu verwenden. Es stellt eine wesentliche Verbesserung gegenüber anderen Sortierverfahren dar, wenn mehr als 40 Daten zu ordnen sind.
Der Kasten zeigt eine Shell-Sortierung am Beispiel eines reduzierten Kartensatzes. Die Kartenreihe wird in Ketten aufgegliedert, deren Anzahl durch die jeweilige Kettenzahl bestimmt wird. Jede Kette wird in sich sortiert, in diesem Beispiel nach dem Vergleichsverfahren. Die Kettenbildung wird so lange fortgesetzt, bis die Kettenzahl kleiner als 1 wird.

Zeichenerklärung
* Teil der Kette 1
\+ Teil der Kette 2
@ Teil der Kette 3
$ Teil der Kette 4

Auf Fehlersuche

Durch sorgfältiges Planen und schrittweises Vorgehen beim Programmieren kann der Zeitaufwand für die Fehlerbeseitigung erheblich verkürzt werden.

„Insektenbekämpfung" = Debugging



Mit zunehmender Programmiererfahrung wird auch das „debugging" (das Beseitigen von Programmierfehlern) geläufiger. Fangen Sie damit an, sich selbst die Aufgabenstellung so klar wie möglich darzulegen. Teilen Sie die Gesamtaufgabe dann in einzelne, logisch in sich geschlossene Teilbereiche, wie Eingabe, Ausgabe, Algorithmen, Datenstruktur, Verarbeitungsprozesse, und betrachten Sie jeden Teilbereich separat.

Beginnen Sie erst mit der Programmeingabe, wenn Sie genau wissen, wie Sie Stück für Stück des Gesamtproblems lösen können! Diese durchdachte Vorgehensweise wird Ihnen viel Zeit beim späteren Ausmerzen von Programmierfehlern ersparen.

Nach dieser Vorarbeit sind Sie in der Lage, strukturierte Programme zu schreiben, die aus einer Anzahl von Subroutinen sowie einem als Gerüst dienenden Hauptprogramm bestehen. In einem solchen Programm sind Fehler leichter zu entdecken. Und es läßt sich eine „Bibliothek" fehlerfreier Subroutinen aufbauen, auf die bei späteren Programmierarbeiten zurückgegriffen werden kann.

Geben Sie den Variablen möglichst aussagekräftige Namen oder Abkürzungen. NET=BRUTT-TAX beispielsweise spricht für sich selbst. Während N=B-T dagegen kaum mehr erkennen läßt, welchen Wert die Variable beinhaltet. Es empfiehlt sich, eine Variablenliste zu führen, die alle im Programm verwendeten Variablen nach Namen und Verwendungszweck enthält. Damit ist die Voraussetzung für eine gewisse Standardisierung geschaffen (beispielsweise können Sie immer einen bestimmten Buchstaben als Variable für Schleifenzähler verwenden). Es sollte ferner vermieden werden, die gleiche Variable für verschiedene Zwecke zu verwenden. Genauso empfiehlt sich, konstante Werte am Anfang des Programms in Variablen zu speichern und später auf diese Variablen zu verweisen. Das Programm wird dadurch schneller und übersichtlicher, und Sie können diese Werte ändern, ohne das Programm danach absuchen zu müssen, wo dieser Wert auftritt.

Trotz des hier beschriebenen Vorgehens sind Programmierfehler nicht ganz zu vermeiden. Es ist darum wichtig, ein Verfahren zur Hand zu haben, mit dessen Hilfe Fehler methodisch aufgefunden werden können. Die bekanntesten „Schnitzer" sind Syntaxfehler – sie können gewöhnlich sofort, wenn der Computer diese anzeigt, beseitigt werden, jedoch kann dies schwierig sein.

Am folgenden Beispiel wollen wir diesen Vorgang einmal besonders deutlich machen. Es handelt sich um einen recht häufigen Fehler:

```
10 PRINT "SCHEINBAR BEDEUTUNGSLOSE
   FEHLER"
20 PRINT "HABEN OFTMALS GROSSE AUS-
   WIRKUNGEN"
```

Solche Zeilen lösen in manchen Fällen eine Fehlermeldung aus, wenn sie nicht als getrennte Zeilen eingegeben wurden. Zeile 10 enthält mehr als 40 Zeichen. Vergessen Sie beim Eingeben, am Ende der Zeile 10 RETURN zu drücken, ehe Sie mit Zeile 20 beginnen, liest der Computer die beiden so perfekt aussehenden Zeilen als eine Zeile.

Falsche Fährte

Fehlermeldungen können aber auch auf die falsche Fährte führen. Die beiden Programmzeilen

```
25 DATA 10.2,34,56.9,0.008, 15.6
30 FOR K=1 TO 5:READ N(K):NEXT K
```

beispielsweise werden wegen eines Fehlers nicht ausgeführt, der scheinbar in Zeile 30 auftritt. Tatsächlich liegt der Fehler aber in Zeile 25: Eine der Nullen wurde fälschlicherweise als Buchstabe O eingegeben.

Codierfehler, die nicht zu Syntaxfehlern führen, sind häufig schwierig zu finden. Hier ist methodisches Vorgehen besonders wichtig. Versuchen Sie zunächst grob herauszufinden, in welchem Programmbereich der Fehler versteckt sein könnte. Bei einem gut strukturierten modularen Programm ist dies gar nicht so schwer. Die Suche wird weiter vereinfacht durch die TRACE-Funktion, die diejenige Programmzeile auf den Bildschirm bringt, die gerade abgearbeitet wird. Fehlt diese Funktion bei Ihrem Computer, können Sie sich durch periodische TRACE-Anweisungen helfen, die Sie in das Programm einbauen (zum Beispiel PRINT "LINE 150" zu Beginn der Zeile 150). Auch der STOP-Befehl ist recht nützlich – mit ihm kann das Programm an bestimmten Stellen angehalten werden, um die Werte kritischer Variablen zu überprüfen. Dies kann direkt durch PRINT oder durch eine Subroutine am Programmende veranlaßt werden:

```
11000 REM DRUCKE DIE VARIABLEN AUS
11100 PRINT "SCORE,SIZE,FLAGS"
11200 PRINT SC;SZ;F1;F2
11300 PRINT "BOARD ARRAY"
11400 FOR K=1 TO 10:PRINT BD$(K):NEXT K
```

Die Folge ist, daß Sie immer dann, wenn das Programm einen STOP-Befehl erreicht, GOTO 11000 eingeben können und der gegenwärtige Variablenstand angezeigt wird. Sie können sogar Änderungen durchführen (zum Beispiel durch Eingeben von SZ=17, RETURN und CONT).

Haben Sie den Fehler bis auf einige bestimmte Zeilen oder eine Variable eingegrenzt, sind Sie nahe am Ziel, aber Vorsicht ist trotzdem geboten. Gehen Sie immer nur um einen Schritt vor und stellen Sie die Auswirkung dieses Schrittes genau fest, ehe Sie weitermachen. Ein besonders fruchtbarer Boden für „Bugs" ist die Anhäufung von Schleifen und Verzweigungen. Hier ist besondere Sorgfalt am Platze und zwar sowohl beim Schreiben als auch bei der Fehlersuche. Betrachten Sie folgendes Beispiel:

```
460 IF SM<0 AND SC< >—1 THEN IF
    SC>0 OR SM=SC—F9 THEN LT=500
470 FOR C1=1 TO LT:FOR C2=LT TO C1
    STEP—1
480 SC=SM+SC*C2
490 NEXT C2:SM=0:NEXT C1
```

Was bedeutet dies alles? Selbst wenn Sie wissen, was geschehen sollte, wissen Sie noch lange nicht, ob das Programm auch läuft. Ein „sicherer" Weg, Fehler einzuschleusen, ist das Einsetzen von Anweisungen innerhalb mehrfach verschachtelter Schleifen. Genauso fehlerträchtig ist es, wenn versäumt wird, beim Schreiben von IF. . .THEN-Anweisungen alle möglichen Konditionen zu beachten. Dies ist besonders dann der Fall, wenn nach IF. . .THEN mehrere Anweisungen folgen. Hier ein Beispiel:

```
655 IF A$=" " THEN GOTO 980:A$=B$
660 PRINT A$
```

Die Anweisung A$=B$ wird nie ausgeführt werden. Entweder ignoriert der Computer A$=" " und das Programm springt auf Zeile 980, oder aber A$< >" ", dann wird der Rest der Zeile 655 nicht beachtet.

Erfahrung ist der beste Lehrer bei der Fehlersuche. Schrittweises Vorgehen und strenge Methodik sind jedoch wertvolle Hilfen.

Ungeziefer

Für Programmierneulinge scheinen Fehler häufig lebendige Wesen zu sein.
Der Fachjargon „debugging" für Fehlersuche wird tatsächlich auf „Ungeziefer" zurückgeführt. Der Ausdruck entstand, als Captain Grace Hopper bei der Suche nach einem Programmfehler einen großen Nachtfalter entdeckte, der sich im elektromechanischen Teil des Computers verfangen hatte und Ursache des „Fehlers" war. Der Ausdruck „debugging" für Fehlersuche war geboren.

Alles auf Band

Die Turing-Maschine ist ein rein theoretisches Hilfsmittel, um herauszufinden, ob ein Problem per Computer gelöst werden kann oder nicht.

Computerwissenschaften haben für die elektronische Datenverarbeitung die gleiche Bedeutung wie die reine Mathematik für den Ingenieurwissenschaftler – es handelt sich um theoretische Verfahren, die jedoch letztendlich die Grundlage für praktische Anwendungen darstellen.

Die Turing-Maschine ist beispielsweise ein theoretisches Denkmodell, entwickelt von Alan Turing für seine Studien mit Algorithmen und der Problem-Berechenbarkeit. Ein Problem läßt sich als „nicht berechenbar" bezeichnen, wenn man beweisen kann, daß es nicht mit der Turing-Maschine gelöst werden kann. Der Mathematiker stellte fest, daß ein solcher Minimalcomputer drei Teile benötigt: einen externen Speicher zur Aufzeichnung und Speicherung von Ein- und Ausgaben, die Mittel, um diesen Speicher zu beschreiben und zu lesen, und eine Kontrolleinheit, um die durchzuführenden Operationen zu bestimmen.

Eine Turing-Maschine wird daher üblicherweise definiert als ein Band (beispielsweise ein Magnetband) und als Kopfmechanismus, der die Symbole lesen oder schreiben kann und der das Bild bewegt. Dabei erhält er von einer Kontrolleinheit Anweisungen, welche Symbole geschrieben werden sollen.

Die Kontrolleinheit enthält ein Ausführungsprogramm, und in dieser Beziehung kann eine Turing-Maschine als speziell für eine bestimmte Anwendung „gebaut" bezeichnet werden, da in der Beschreibung keine Möglichkeit vorgesehen ist, ein Programm zu laden oder zu ändern. Dabei wird absichtlich die Schreibweise „gebaut" verwendet, weil die wenigen Turing-Maschinen, die wirklich konstruiert worden sind, ausschließlich zu Lehrzwecken dienten. Es ist jedoch ziemlich einfach, ein BASIC-Programm zu schreiben, das die Arbeit einer Turing-Maschine auf einem Heimcomputer simulieren kann.

Das Kontrollprogramm in einer Turing-Maschine besteht aus einer Reihe von „Fünf-Tupeln". Dies sind Ausdrücke, die fünf Elemente enthalten. Welcher Tupel in einem bestimmten Zustand ausgeführt wird, hängt von zwei Faktoren ab: Dem Symbol in dem Quadrat, das sich gerade unter dem Bandkopf befindet, und dem „Zustand" der Maschine. Dieser Zustand ist ein völlig willkürliches Kriterium: Sie können festlegen, daß die Maschine im Zustand S(A) startet und anhält, wenn sie den besonderen Zustand H erreicht und die Berechnung beendet ist. In der Zwischenzeit wird sich der Zustand gemäß den Anweisungen der Fünf-Tupel viele Male ändern.

Die fünf Elemente jedes Tupels sind:

1) Der augenblickliche Zustand der Maschine.
2) Das Symbol in dem Bandquadrat unter dem Kopf.
3) Das Symbol, das in dieses Quadrat geschrieben werden soll (das gleiche wie 2, wenn keine Änderung der Daten verlangt wird).
4) Der Zustand, in den die Maschine als nächstes übergehen sollte.
5) Die Richtung, in die der Bandkopf bewegt werden soll – links oder rechts.

Der Tupel (S(A), 5, 3, S(B), R) zum Beispiel wird immer dann ausgeführt werden, wenn die Maschine im Zustand S(A) ist und der Bandkopf eine 5 liest. Die 5 wird dann zu einer 3 verändert. Die Maschine geht über in den Zustand S(B) und der Kopf wird um ein Quadrat nach rechts bewegt.

Modell-Entwurf

Um eine Turing-Maschine zu entwerfen, die eine konkrete Aufgabe erfüllen soll, müssen Sie das Format angeben, in dem die Eingabedaten der Maschine auf dem Band geliefert werden, das Format der Ausgabedaten auf dem Band nach Beendigung der Rechnung (wenn die Maschine, in unserem Beispiel, im Zustand H ist) und die Fünf-Tupel, die erforderlich sind, um den Algorithmus auszuführen.

Im Kasten sehen Sie eine Turing-Maschine, die die AND-Funktion berechnen soll. Sie müssen die zwei Eingabebits (jedes eine 1 oder eine 0) in aufeinanderfolgende Quadrate schreiben, gefolgt durch ein Fragezeichen-Symbol, das durch die Antwort (wieder eine 1 oder eine 0, abhängig von der Eingabe) ersetzt wird. Zur Dekoration wurde ein Sternchen-Symbol an beiden Enden des Datenblocks angefügt. Sie starten die Maschine im Zustand S(A) auf dem linken Sternchen, diese stoppt auf dem rechten.

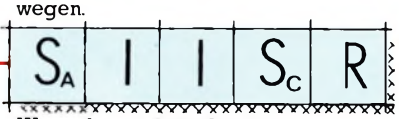
Insgesamt zehn Fünf-Tupel werden benötigt, um diese Maschine zu definieren, obwohl, wie Sie in dem abgearbeiteten Beispiel sehen können, für jeden Lauf nur fünf gebraucht werden. Wenn Sie mit derselben Maschine einen Versuch mit 0 AND 1 machen, werden Sie feststellen, daß ein anderer Teil von Tupeln aus den zehn ausgewählt wird.

Turing-Maschine

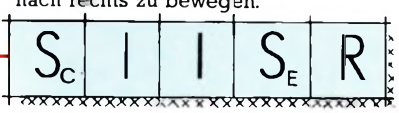
Dieses Beispiel zeigt eine Turing-Maschine zur Berechnung der AND-Funktion. Die beiden Eingabebits werden in aufeinanderfolgende Quadrate geschrieben, gefolgt von einem Fragezeichen, das durch das Ergebnis ersetzt wird. Zwei Sternchen werden als Begrenzer an die Enden des Datenblocks plaziert. Die zehn Fünf-Tupel unten definieren die Operationen der Maschine, obwohl für jedes bearbeitete Beispiel (in diesem Fall 1 AND 1) nur fünf von zehn gebraucht werden.

S_A	*	*	S_A	R
S_A	0	0	S_B	R
S_A	1	1	S_C	R
S_B	0	0	S_D	R
S_B	1	1	S_D	R
S_C	0	0	S_D	R
S_C	1	1	S_E	R
S_D	?	0	S_F	R
S_E	?	1	S_F	R
S_F	*	*	H	R

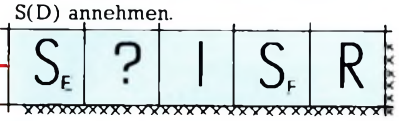
Die Maschine startet im Zustand S(A) mit dem Kopf über dem linken Stern. Der Effekt des Tupels ist, den Bandkopf einen Schritt nach rechts zu bewegen.



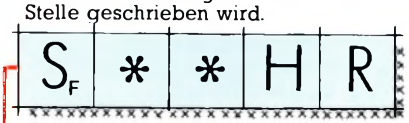
Wenn das nächste Quadrat eine 1 enthält, wird dieser Tupel angewählt: Die Maschine geht in den Zustand S(C) und wird angewiesen, das Band nach rechts zu bewegen.



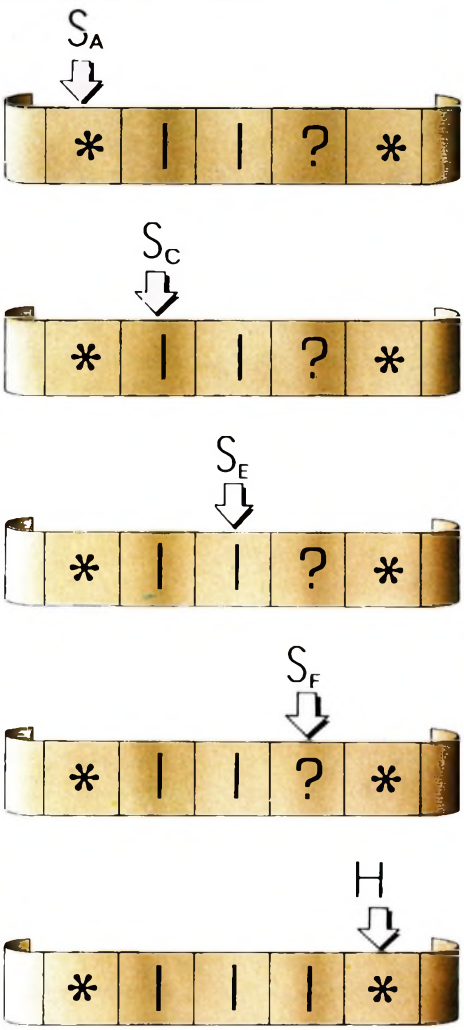
Da die Maschine im Zustand S(C) ist, bewirkt eine 1 im zweiten Quadrat den Zustand S(E). In allen anderen Fällen würde die Maschine Zustand S(D) annehmen.

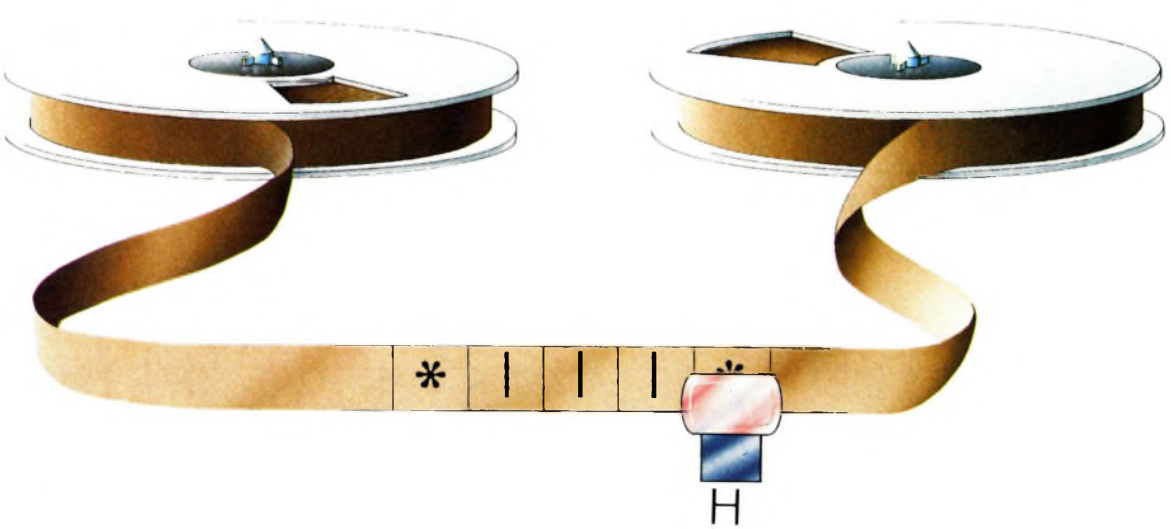


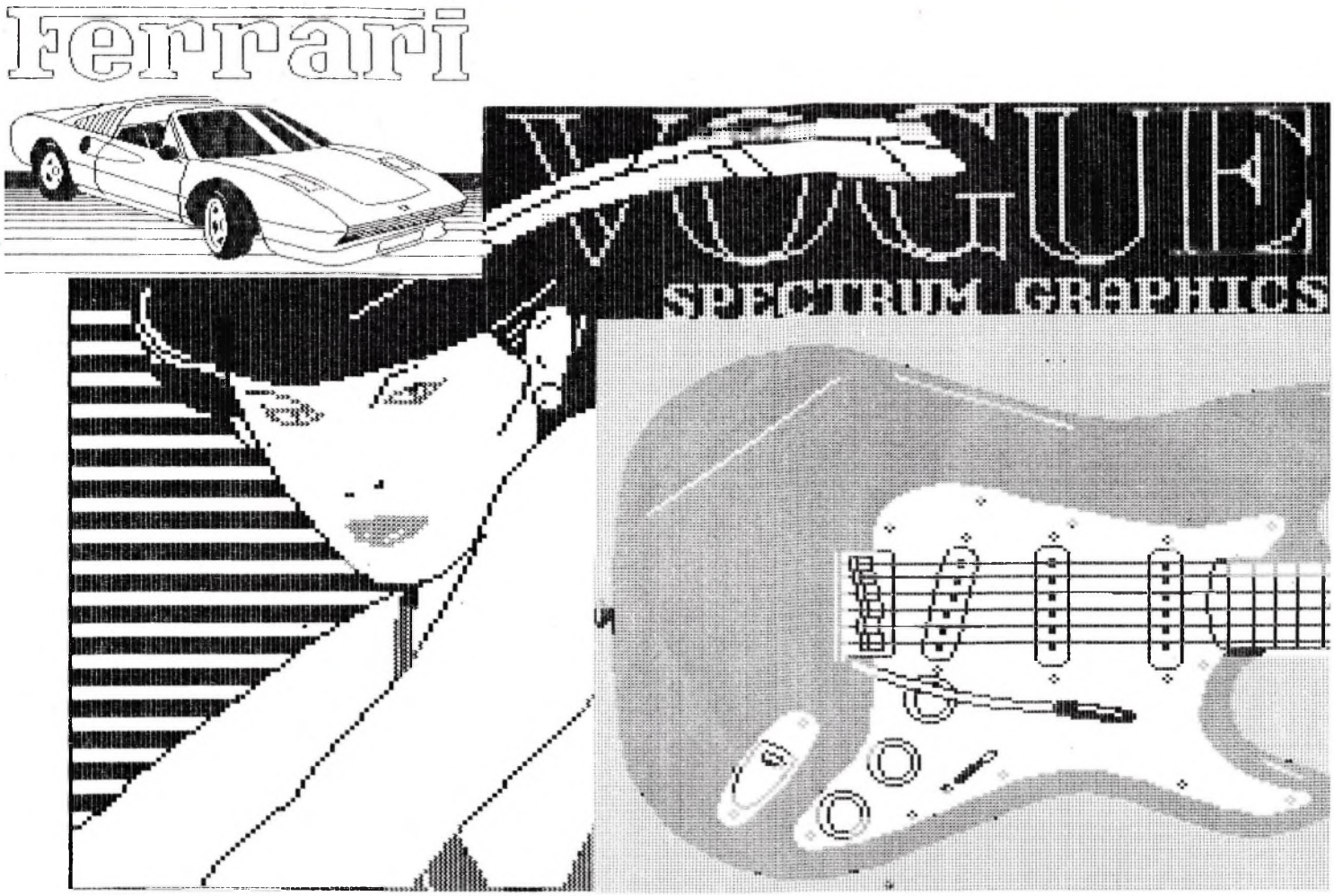
Wenn ein Fragezeichen gelesen wird, bestimmt der Zustand der Maschine, S(E) oder S(D), ob eine 0 oder eine 1 als Ergebnis an seine Stelle geschrieben wird.



Die Maschine geht nur über dem zweiten Sternchen in den Haltezustand H. Sie können die Operation auf dem Papier für 1 AND 0, 0 AND 1 und 0 AND 0 ausprobieren.







Dies sind Beispiele, welche Arten von Ausdrucken bestimmte Matrixdrucker herstellen können. Jede Nadel des Druckkopfes wird einzeln gesteuert, so daß zufriedenstellende Bildmuster gedruckt werden können. Die hier gezeigten Beispiele entstanden unter Verwendung der „Paintbox" von Print'n'Plotter Products.

Punkte, Punkte

Die Fehlersuche wird mit einer gedruckten Ausgabe der Listings um vieles leichter, und die Textverarbeitung ist ohne Drucker undenkbar.

Der Kauf eines Druckers ist, aufgrund des vielfältigen Angebotes, ebenso schwierig wie der eines Computers. Die richtige Wahl ist für den unerfahrenen Käufer dementsprechend schwierig. Die erste Entscheidung, die zu treffen ist, betrifft die Art des Druckers. Soll es ein Matrixdrucker, ein Typenraddrucker, ein Thermo- oder Tintenstrahldrucker sein? Typenraddrucker liefern die beste Schriftqualität, liegen dafür aber auch in den gehobeneren Preisklassen. Sie eignen sich gut zur Textverarbeitung. Der Matrixdrucker dagegen arbeitet schneller, ist ideal für Listingausdruck, grafikfähig und preisgünstiger.

Einfache Matrixdrucker sind schon ab 500 Mark zu bekommen, komfortablere Geräte dagegen können 3000 Mark und mehr kosten. Wesentliche Beurteilungs-Merkmale sind die Druckgeschwindigkeit und die Qualität des erzeugten Schriftbildes. Dazu kommen noch eventuelle Extras wie Proportionalschrift, verschiedene Schriftgrößen usw. Bei der Proportionalschrift wird die Breite des einzelnen Buchstabens berücksichtigt und das nächste Zeichen entsprechend plaziert, so daß ein gleichmäßiges Schriftbild entsteht: Ein „i" braucht zum Beispiel weniger Platz als ein „m". Ob Ihnen diese Besonderheiten einen höheren Kaufpreis wert sind, müssen Sie selbst entscheiden.

Die Druckgeschwindigkeit beeinflußt die Zeitspanne, in der der Computer für den Ausdruck in Anspruch genommen wird. Währenddessen kann der Rechner nicht für andere Aufgaben genutzt werden. Die Druckgeschwindigkeit wird in Z/s (Zeichen je Sekunde, in Englisch cps = characters per second) angegeben. Bei einem Programm, für das ein mit 200 cps arbeitender Drucker beispielsweise eine Minute benötigt, würde ein einfacherer Drucker mit 30 cps mehr als sechs Minuten arbeiten müssen – und den Computer auch so

lange blockieren. Diese unfreiwillige Arbeitspause läßt sich mit Hilfe eines Zwischenspeichers vermeiden. Ein solcher Buffer ist nicht anderes als eine Platine mit RAM-Chips, die zwischen Computer und Drucker geschaltet ist. Der Computer gibt die auszudruckenden Daten in der ihm eigenen hohen Geschwindigkeit in den Zwischenspeicher und ist dann für andere Aufgaben frei. Der Zwischenspeicher („Buffer") übermittelt anschließend die auszudruckenden Zeichen in einer dem Drucker angepaßten Geschwindigkeit weiter. In komfortablen Druckern ist ein solcher Speicher bereits eingebaut.

Die Geschwindigkeit

Mit den vom Hersteller angegebenen Werten für die Druckgeschwindigkeit ist es wie mit den Angaben über den Benzinverbrauch bei den Autos: Sie sollten nicht wörtlich genommen werden, denn sie beruhen auf idealen Bedingungen und haben mit der Realität nur selten etwas gemein. Beispielsweise wird die Druckgeschwindigkeit beim Drucken einer Zeile mit immer dem gleichen Zeichen gemessen. Unterschiedliche Schriftzeichen, Wagenrücklauf und Zeilenvorschub setzen natürlich die Druckgeschwindigkeit herab. In der Praxis heißt dies, daß ein Drucker, dessen „Spitzengeschwindigkeit" mit 160 cps angegeben ist, tatsächlich kaum mehr als 100 Zeichen je Sekunde erreicht.

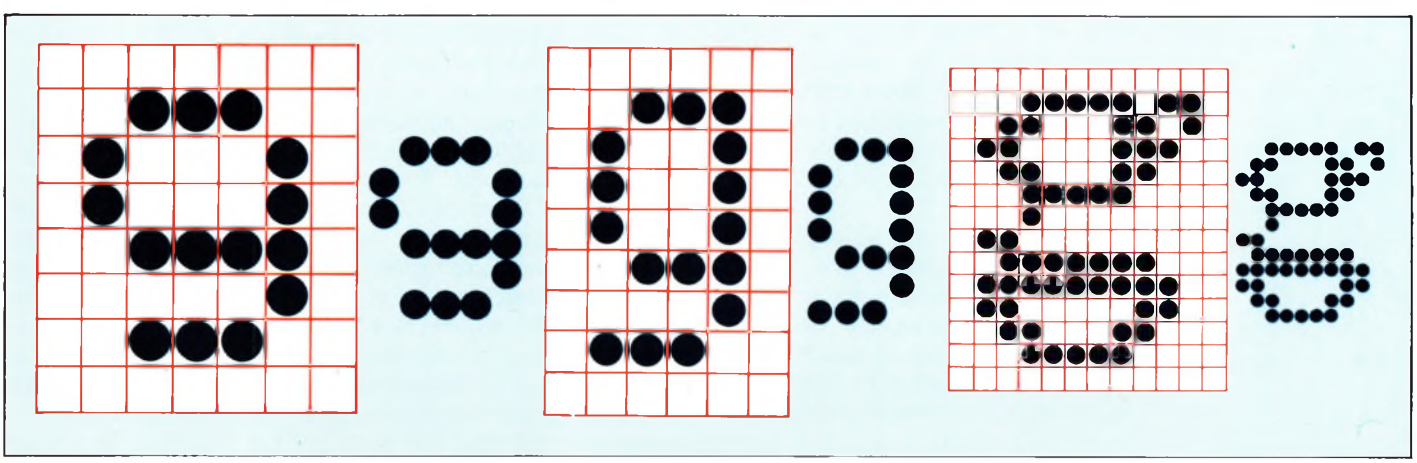
Die Druckqualität der Matrixdrucker ist unterschiedlich. Sie hängt davon ab, wie viele Nadeln im Druckkopf zur Verfügung stehen. Es gibt Modelle mit sieben Nadeln, aber auch solche mit 16 oder mehr. Der Commodore-Drukker zum Beispiel bildet die Zeichen in Form einer 7x6-Matrix. Der PW1080 von Canon dagegen hat 16 Stifte und erzeugt für jedes Zeichen eine 16x23-Matrix. Da gedruckte Schriftzeichen aus einzelnen Punkten bestehen, wird die Schrift um so klarer sein, je mehr Punkte zum Bilden eines Zeichens verwendet werden. Geht es um reine Programmlistings, kann auf eine hochwertige Druckqualität verzichtet werden. Bei der Textverarbeitung dagegen ist ein gutes Schriftbild meist sehr wichtig.

Der Matrixdrucker ist im Grunde genommen nichts weiter als ein Einzweck-Microcomputer; denn er besitzt einen Microprozessor und arbeitet mit ROM und RAM. Aufgrund dieser technischen Voraussetzungen ist es möglich, den Drucker zu programmieren. Dies geschieht durch bestimmte Steuerzeichen oder durch Verstellen des DIL-Schalters (Dual-in-Line) im oder am Gehäuse. Auf diese Weise kann beispielsweise der standardmäßige ASCII-Zeichensatz, der im Drucker gespeichert ist, umdefiniert werden.

Gesperrte, halbfette oder fette Schrift und unterschiedliche Zeilenabstände sind weitere Extras. Der FX80 von Epson beispielsweise besitzt zahlreiche solcher Besonderheiten, wie kursive Schriftzeichen, automatisches Unterstreichen oder Proportionaldruck, die ihn zu einem der vielseitigsten Drucker machen. Die Epson-Drucker sind mittlerweile zu einer Art „industriellem Standard" geworden. Das hat dazu geführt, daß viel Software so entwickelt wird, daß sie nur mit einem Epson-Drucker arbeiten kann. Dies ist besonders interessant, da die Drucker der verschiedenen Hersteller keineswegs kompatibel sind.

Zu den weiteren Aspekten, die beim Kauf eines Druckers bedacht werden sollten, gehört die Zuverlässigkeit. Ein „Billig-Drucker" mag sich zwar für das gelegentliche Ausdrucken von Programmen als brauchbar erweisen, ist aber denkbar ungeeignet für den Dauereinsatz im Büro. Ein anderer Faktor ist der Lärm. Sollten Sie ein ausgesprochener „Nachtarbeiter" sein, kann ein laut hämmernder Drucker zur Katastrophe werden. Auch die Papierzuführung ist der Betrachtung wert. Zwar können alle Matrixdrucker Endlospapier verarbeiten, für Einzelblätter wird jedoch ein Rolleneinzug benötigt.

Ganz zum Schluß sollte die wichtigste Frage nicht vergessen werden: „Paßt der Drucker auch an meinen Computer?" Die meisten Matrixdrucker haben entweder eine Centronics- oder eine RS232-Schnittstelle. Bei Anschlußschwierigkeiten hilft unter Umständen eine Adapter-Schnittstelle.

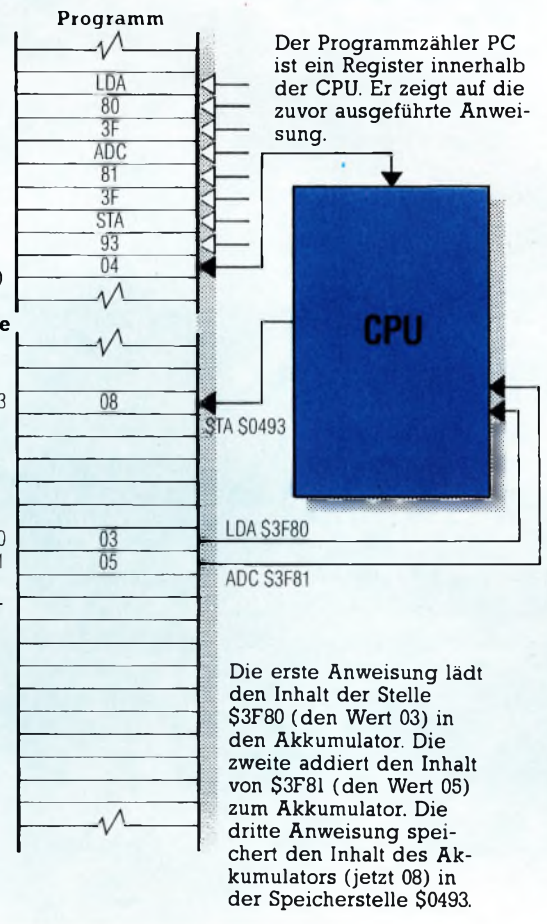


Diese Druckmuster zeigen die Qualitätsunterschiede zwischen verschiedenen Matrixdrukkern. Die Druckqualität hängt im wesentlichen von der Anzahl der „Nadeln" ab, die sich im Druckkopf befinden – je mehr Druckstifte, desto besser die Schriftzeichen.

Maschinensprache

Der Sprung von BASIC zur Maschinensprache erfordert ein völliges Umdenken. Als Belohnung winkt jedoch eine enorme Steigerung der Ausführungsgeschwindigkeit und Leistungsfähigkeit der Programme.

Schritt für Schritt



Das Maschinensprache-Programm ist in einem Speicherbereich abgelegt, die zu verarbeitenden Daten dagegen können in einem anderen Bereich gespeichert sein. Die Operanden (z. B. $3F80) belegen jeweils zwei Bytes, wobei das niedrigere Byte ($80) vor dem höherwertigen ($3F) steht.

Der Programmzähler PC ist ein Register innerhalb der CPU. Er zeigt auf die zuvor ausgeführte Anweisung.

Jedes Maschinensprache-Programm besteht aus einfachen Operationen, die den Inhalt von Speicherstellen in die internen CPU-Register übertragen, sie verarbeiten und das Ergebnis erneut speichern. Hier wird ein Programm dargestellt, das den Inhalt von zwei Speicherstellen addiert und das Ergebnis in einer dritten ablegt.

Die erste Anweisung lädt den Inhalt der Stelle $3F80 (den Wert 03) in den Akkumulator. Die zweite addiert den Inhalt von $3F81 (den Wert 05) zum Akkumulator. Die dritte Anweisung speichert den Inhalt des Akkumulators (jetzt 08) in der Speicherstelle $0493.

Bisher hat sich das ganze Programmieren um die Sprachen BASIC und LOGO gedreht, weil sie vielseitig und leicht zu handhaben sind. Diese Sprachen haben aber auch Ihre Grenzen. Je umfangreicher Ihre Programme werden, desto schneller erreichen Sie den Punkt, an dem Ihnen die Grafiken zu langsam über den Bildschirm wandern und die PEEKs und POKEs nur noch schwer zu überschauen sind.

Die Maschinensprache dagegen erlaubt Ihnen nahezu unbegrenzte Programmiermöglichkeiten und vermittelt das Gefühl eines „Geschwindigkeitsrausches". Daß nur wenige Besitzer eines Heimcomputers diese Umstellung wagen, liegt an zwei Problemen: Die Programmierung in Maschinensprache ist sehr arbeitsintensiv, und die Logik des Maschinencodes hat kaum etwas mit einer anderen Programmiersprache gemein. Es lohnt sich jedoch, diese Logik zu lernen.

Die Zentraleinheit, Herz eines jeden Computers, versteht nur Maschinensprache. Außerdem kann sie nur ganz simple Funktionen ausführen (beispielsweise Addieren, nicht aber Multiplizieren).

Ein in Maschinensprache geschriebenes Programm besteht aus diesem Grund aus einer Vielzahl von Anweisungen, wobei jede Funktion „von Hand" aus einfachen Befehlen aufgebaut werden muß. Programmieren in Maschinensprache heißt, individuelle Bits und Bytes im Speicher unter Verwendung einfacher logischer Operationen wie AND, OR, NOT und elementarer Binärarithmetik zu manipulieren.

Dies ist der eine Grund, warum das Programmieren in Maschinensprache so langsam vonstatten geht. Der andere ist die Notwendigkeit, daß der Programmierer genau wissen muß, wo welche Daten im Speicher zu finden sind. Tritt in BASIC der Befehl LET A=5 auf, ist es Sache des BASIC-Interpreters, einen Platz im Speicher zu finden, wo die Variable abgelegt werden kann. Er muß sich ebenfalls immer an diese Stelle „erinnern", wenn die Variable A verlangt wird. Beim Programmieren in Maschinensprache dagegen muß jede zu speichernde Information eine Adresse erhalten. Und der Programmierer selbst muß dafür sorgen, daß es nicht zum unbeabsichtigten Überschreiben von Daten kommt.

Nun zu den Bestandteilen der Maschinensprache. Alle folgenden Beispiele beziehen sich auf eine 8-Bit-CPU wie die Z80 oder die 6502 (16-Bit-CPUs arbeiten ähnlich, jedoch mit doppelter Bit-Zahl). Die Zentraleinheit ist durch zwei Bus-Leitungen mit dem Speicher verbunden: durch den Adreß-Bus und den Daten-Bus. Es gibt außerdem den Kontroll-Bus zur Übertragung der Zeitsteuersignale. Er wird jedoch nicht vom Programmierer benutzt.

Über den Adreß-Bus ist die CPU in der Lage, mit Hilfe der verschiedenen Bit-Muster jede der 65536 Speicherstellen im Speicherplan anzuwählen. In einem Heimcomputer bestehen diese Speicherstellen aus RAM-, ROM- oder auch besonderen Ein-/Ausgabe-Chips. Will die Zentraleinheit eine Speicherstelle lesen (durch ein Lese-/Schreib-Signal zeigt der Kontroll-Bus die gewünschte Funktion an), wird

der Inhalt des betreffenden Bytes in Form eines 8-Bit-Musters auf den Daten-Bus „gepackt". In gleicher Weise kann die CPU ein solches Muster in eine ausgewählte Speicherstelle schreiben. Da die Zentraleinheit jedoch nicht „weiß", wie die RAMs und ROMs verteilt sind, muß der Programmierer die richtige Adresse herausfinden.

Innerhalb der CPU gibt es eine Handvoll „Register". Das sind individuelle Speicher, die zum zeitweiligen Abspeichern von Ergebnissen, logischen Operationen und binären Funktionen benötigt werden. Die meisten Register entsprechen einer 1-Byte-Speicherstelle. Es gibt aber auch 16-Bit-Register wie den Programmzähler PC (Program Counter), der immer die Adresse des gerade auszuführenden Maschinensprache-Befehls enthält. Man könnte dies mit der Zeilennummer in BASIC vergleichen.

Ein anderes wichtiges Register (diesmal mit 8 Bit) ist der „Akkumulator". Er kann, wie der Name sagt, Ergebnisse „ansammeln". Bytes können hinzuaddiert oder abgezogen werden. Es ist tatsächlich das einzige Register, das eine Arithmetik ausführen kann. Ein einfaches Maschinensprache-Programm könnte so beschrieben werden:

1) Lade den Akkumulator mit dem Inhalt der Speicherstelle $3F80. (Adressen in Maschinensprache werden gewöhnlich hexadezimal geschrieben. Hexadezimalwerte werden üblicherweise durch das Zeichen $ gekennzeichnet.)
2) Addiere zum Akkumulator den Inhalt der Speicherstelle $3F81. Ist das Ergebnis größer als ein Byte, tritt das "carry bit" (Übertrag-Bit) in Aktion.
3) Speichere den neuen Inhalt des Akkumulators (das Ergebnis) in die Speicherstelle $0493.

Jede dieser Anweisungen entspricht einem Maschinensprache-Befehl, und das Programm würde normalerweise so geschrieben:

```
LDA $3F80  (LoaD Accumulator; Lade Akkumulator)
ADC $3F81  (ADd with Carry; Addiere mit Übertrag)
STA $0493  (STore Accumulator; Speichere Akkumulator-Inhalt)
```

Die in Klammern stehenden Anmerkungen sind, wie die REMark-Kommentare in BASIC, ohne Auswirkung auf den Programmablauf. Jede Zeile beginnt mit dem „Opcode" (Operationscode), der die Art der Operation angibt. Es folgt der „Operand" mit Einzelheiten über die benötigten Daten. Normalerweise kann eine CPU mehrere Dutzend Opcodes verarbeiten, wobei eine Operation jeweils nur ein Byte belegt.

Ein Opcode kann daher nur Werte zwischen 0 und 255 ($00 bis $FF hexadezimal) annehmen. Aus Gründen der Lesbarkeit wird jedoch bei der Programmerstellung üblicherweise eine sinngemäße Abkürzung aus drei Buchstaben zur Hilfe genommen, beispielsweise LDA für LoaD Accumulator.

Jeder der drei oben genannten Operanden enthält eine Hex-Zahl zwischen $0000 und $FFFF und belegt damit bis zu zwei Speicherstellen. Manche Operanden sind jedoch nur ein Byte lang, und einige Opcodes haben überhaupt keine Operanden. Das hier gezeigte kurze Programm würde demnach nur neun Bytes beanspruchen, ausgenommen die drei Speicherstellen $3F80, $3F81 und $0493, die adressiert werden. Für dieses einfache Beispiel könnte auch das folgende BASIC-Programm verwendet werden. Es würde jedoch 50 Bytes belegen und hundertmal langsamer laufen, weil der Interpreter erst einmal alles übersetzen muß:

```
10 A=PEEK (16256)
20 A=A+PEEK (16257)
30 POKE 1171,A
```

Lichterketten

Die Idee für die riesigen Tafeln mit blinkenden Lichtern, wie sie in Filmen an Computern zu sehen sind, resultiert aus den Fronttafeln, die früher bei vielen Microcomputern zu finden waren. Sie bestanden aus Reihen von Lampen und Schaltern für die Adreß- und Datenleitungen der CPU. Als Tastaturen noch nicht angeschlossen werden konnten, mußten alle Maschinensprache-Programme auf diese Weise eingegeben werden.

LDA — **LoaD Accumulator** überträgt den Inhalt einer einzelnen Speicherstelle (Byte) in das interne Akkumulator-Register.

STA — **STore Accumulator** führt den gegenteiligen Vorgang zu LDA aus.

ADC — **ADd with Carry** addiert den Inhalt einer Speicherstelle zum Inhalt des Akkumulators und erzeugt, wenn notwendig, ein Übertrag-Bit.

SBC — **SuBtract with Carry** ist die Gegenfunktion von ADC.

JMP — **JuMP** überträgt eine Operation zu einer neuen Stelle (ähnlich wie GOTO in BASIC).

Links finden Sie einige Opcode-Beispiele, die von den meisten Microprozessoren verarbeitet werden können.

„Verhexte" Befehle

In der Fortsetzung unserer Serie über den Maschinencode untersuchen wir die vielfältigen Formen, in denen ein Programm eingegeben werden kann – von binär bis Assemblersprache.

Anfänger auf dem Gebiet des Maschinencodes haben oft Probleme mit den vielen unterschiedlichen Formaten, in denen die Programme geschrieben werden können. Obwohl im Speicher des Computers alle Daten über Binärzahlen (mit der Länge von je acht Bit) abgelegt werden, sind diese Zahlenreihen in gedruckter Form sehr unübersichtlich, schwer zu behalten und anfällig für Tippfehler. Für ihre Ein- und Ausgabe wird daher die hexadezimale Form verwendet. Sie hat den Vorteil, daß der Inhalt eines Bytes als zweistellige Zahl ausgedrückt werden kann und daß sich jede Adresse des Arbeitsspeichers (0–65535) vierstellig darstellen läßt.

Damit sie sich von einer Dezimalzahl unterscheidet, wird eine Hexadezimalzahl normalerweise mit einem vorangehenden „$"-Zeichen geschrieben. Bei einigen „Opcodes" besteht außerdem die Notwendigkeit, einen aus zwei Bytes bestehenden Operanden (z. B. LDA $3F80) in umgekehrter Folge einzugeben. D. h., es wird zunächst das niederwertige Byte eingegeben und dann das höherwertige. In unserem Beispiel ist die Eingabereihenfolge AD (die hexadezimale Darstellung des Opcodes für LDA auf dem 6502), gefolgt von 80 und 3F.

Normalerweise wird ein Maschinenprogramm als „Hex Dump" gedruckt, in Form einer langen Liste mit zweistelligen Hexadezimalwerten. Dabei muß die Speicherstelle (in dezimal oder in hex) angegeben werden, die mit dem ersten Hexadezimalwert belegt werden soll. Der zweite Hexadezimalwert wird dann in die nächsthöhere Speicherstelle geladen usw. Der Ladevorgang läßt sich mit dem BASIC-Befehl POKE ausführen. Die Anfangsadresse wird dabei auf $1000 (dezimal 4096) gesetzt, und die hexadezimale Liste sieht folgendermaßen aus:

```
AD (dezimal 173)
80 (dezimal 128)
3F (dezimal 63)
```

Die folgenden drei BASIC-Befehle laden das Programm:

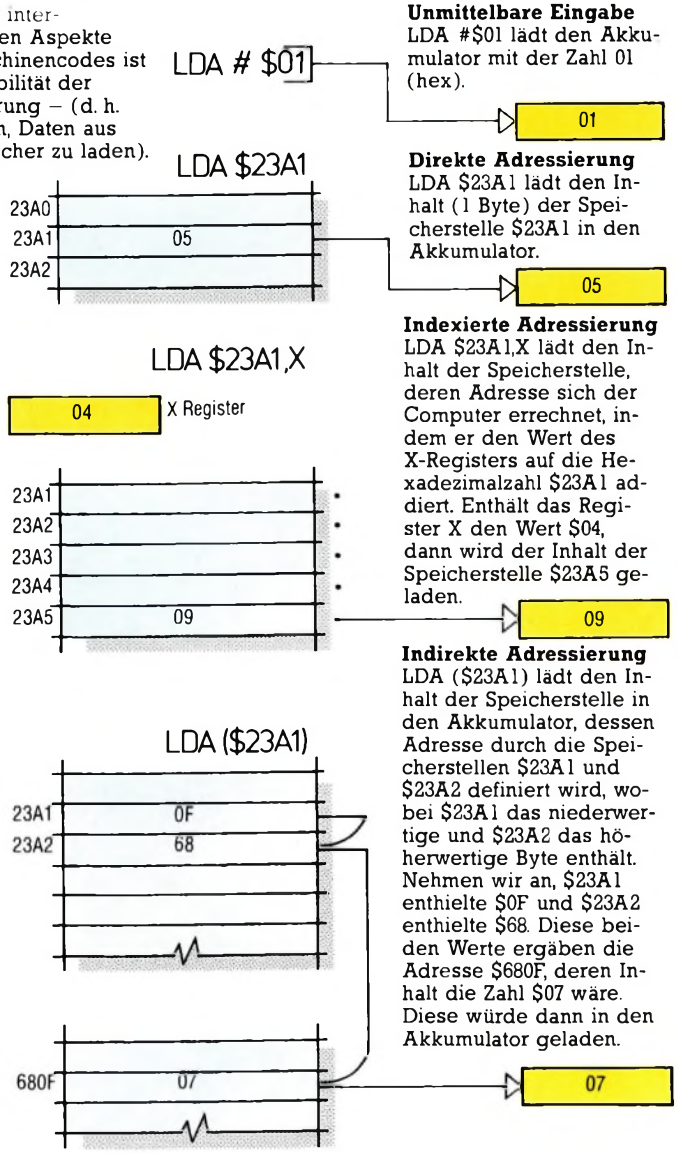
```
POKE 4096, 173
POKE 4097, 128
POKE 4098, 63
```

Hierbei müssen alle hexadezimalen Werte in Dezimalzahlen umgewandelt werden, bevor sie in dem POKE-Befehl eingesetzt werden können.

Längere Maschinenprogramme lassen sich über ein kurzes BASIC-Programm – ein „Lademodul für Maschinencode" – eingeben. Dabei wird zunächst die Anfangsadresse des Programms angegeben und dann jeder einzelne Hexadezimalwert. Nach den Eingaben wandelt das Lademodul den Hexwert in eine Dezimalzahl um und setzt diese mit POKE an die näch-

Adressierungsarten

Einer der interessantesten Aspekte des Maschinencodes ist die Flexibilität der Adressierung – (d. h. Methoden, Daten aus dem Speicher zu laden).



Unmittelbare Eingabe
LDA #$01 lädt den Akkumulator mit der Zahl 01 (hex).

Direkte Adressierung
LDA $23A1 lädt den Inhalt (1 Byte) der Speicherstelle $23A1 in den Akkumulator.

Indexierte Adressierung
LDA $23A1,X lädt den Inhalt der Speicherstelle, deren Adresse sich der Computer errechnet, indem er den Wert des X-Registers auf die Hexadezimalzahl $23A1 addiert. Enthält das Register X den Wert $04, dann wird der Inhalt der Speicherstelle $23A5 geladen.

Indirekte Adressierung
LDA ($23A1) lädt den Inhalt der Speicherstelle in den Akkumulator, dessen Adresse durch die Speicherstellen $23A1 und $23A2 definiert wird, wobei $23A1 das niederwertige und $23A2 das höherwertige Byte enthält. Nehmen wir an, $23A1 enthielte $0F und $23A2 enthielte $68. Diese beiden Werte ergäben die Adresse $680F, deren Inhalt die Zahl $07 wäre. Diese würde dann in den Akkumulator geladen.

Bits und Mnemonics

Programme im Maschinencode können auf mehrere Arten geschrieben werden. Normalerweise wird dafür die Assemblersprache eingesetzt, in der die Opcodes (Maschinenbefehle) als Mnemonics eingegeben und symbolische Bezeichnungen (Labels) verwendet werden können. Beispiel:

```
LDA GEWICHT
ADC BENZIN
STA GEWICHT
```

Für symbolische Bezeichnungen müssen allerdings die Adressen angegeben werden wie z. B.:

```
BENZIN = $03EE
GEWICHT = $031F
```

Ein Assembler-Programmpaket übersetzt diese Angaben in einen Hex Dump und speichert sie auf einer Diskette. Der „Pseudoassembler" ist nicht so einfach zu lesen, kann aber von einem „Spot Assembler" ohne ein Diskettenlaufwerk unmittelbar umgesetzt werden.

```
LDA $031F
ADC $03EE
STA $031F
```

Ein Hex Dump (Listing im Maschinencode) besteht aus einer Anfangsadresse (links) und einer Folge von zweistelligen Hexadezimalwerten, die dem Inhalt des Arbeitsspeichers entsprechen. Die Operanden (z. B. $031F) werden dabei in umgekehrter Reihenfolge gespeichert (1F03), und die Opcodes werden durch die entsprechenden Hexadezimalwerte ersetzt.

```
19C4 AD 1F 03 6D EE 03 8D 1F 03
```

ste erreichbare Speicherstelle.

Die meisten Computer verfügen über Befehle, mit denen das Gerät den BASIC-Modus verlassen und von einer bestimmten Speicherstelle an mit der Ausführung des Maschinencodes beginnen kann. So bedeutet der Befehl SYS 4096 (RETURN): „Übergib die Steuerung dem System, und beginne mit der Ausführung der Instruktion in Speicherstelle 4096". Eine weitere Möglichkeit ist CALL $E651. Hier wird eine Routine im Maschinencode aufgerufen, die an der Speicherstelle (hex) E651 anfängt.

BASIC-Beschleunigung

Bei Eingabe dieser Befehle führt das Gerät die Maschinenroutine aus. Ist diese fehlerfrei und verfügt sie an ihrem Ende über einen korrekten Abschluß, wird die Steuerung danach wieder an das BASIC zurückgegeben. Mit Maschinenroutinen lassen sich einzelne Funktionen eines BASIC-Programms erheblich beschleunigen, wobei die gleichen Routinen durchaus von mehreren Stellen aus aufgerufen werden können.

Leider werden Fehler im Maschinencode nicht mit einem hilfreichen „SYNTAX ERROR" angezeigt, sondern lassen das System „abstürzen": Das Gerät unterbricht die Arbeit und nimmt keine Eingaben mehr entgegen. Dieser Vorgang schadet zwar dem Computer nicht, Ihr Programm ist allerdings gelöscht und muß neu eingegeben werden. Mit dem Maschinencode läßt sich daher nicht so bequem experimentieren wie mit einem BASIC-Programm.

Es gibt jedoch ein Hilfsprogramm, das große Unterstützung bietet: der „Monitor für Maschinencode" (nicht zu verwechseln mit dem Bildschirmmonitor). Einige Hersteller haben den Monitor als ROM bereits im Gerät integriert, normalerweise ist er jedoch auf einer Cassette oder als Cartridge erhältlich. Ein Monitor für Maschinencode ist ein einfaches Betriebssystem, das den Inhalt des Arbeitsspeichers in Blöcken auf dem Bildschirm anzeigt. Mit dem Monitor lassen sich Speicherstellen leicht verändern oder überschreiben. Er ist eine der schnellsten Methoden, Hexadezimalcode einzugeben. Mit diesem Programm läßt sich außerdem Maschinencode ohne ein BASIC-Lademodul auf Cassette speichern oder davon einlesen.

Hex Dumps sind zwar eine bequeme Methode zur Darstellung des Maschinencodes, lassen sich aber nur schwer lesen. Es ist so gut wie unmöglich, Opcode und Operanden voneinander zu unterscheiden, wenn man nicht die hexadezimalen Äquivalente aller Opcodes im Gedächtnis hat. Maschinenprogramme werden daher mit Kürzeln geschrieben, die aus drei Buchstaben bestehen – sogenannten „Mnemonics" – und anhand der Befehlstabelle des Prozessors in hexadezimale Form übersetzt.

Die Assemblersprache ist eine Programmiermethode für Maschinencode, bei der nicht nur die Mnemonics des Opcodes verwendet werden, sondern anstelle von Hexadezimalzahlen auch Namen (oder symbolische Bezeichnungen) als Operanden eingesetzt werden können. Legt ein Spiel z. B. in der Speicherstelle $07B2 die Anzahl der verschossenen Raketen ab, dann kann dieser Wert mit folgendem Befehl in den Akkumulator geladen werden:

```
LDA RAKETEN
```

Am Anfang des Programmes muß natürlich angegeben werden, daß die Speicherstelle für RAKETEN = $07B2 ist und daß dieser Speicher beim Start auf den Wert $09 (neun Raketen) gesetzt wird.

Ist das Programm in der Assemblersprache (dem sogenannten Source Code) abgeschlossen, wird es von einem Hilfsprogramm, dem „Assembler", bearbeitet. Der Assembler ersetzt die Mnemonics und symbolischen Bezeichnungen jedes Befehls durch ihre hexadezimale Entsprechung und stellt eine neue Version des Programms her, den „Object Code". Dieser Code kann in den Arbeitsspeicher eines Computers geladen und anschließend gestartet werden.

Opcode

Einige Opcodes, die für einen Microprozessor typisch sind:

Jump SubRoutine (Springe zur Unterroutine)
Diese Funktion entspricht dem BASIC-Befehl GOSUB. JSR $354D verändert den Wert des Programmzählers (PC-Register), so daß der Code von der Speicherstelle $354D an ausgeführt wird.

ReTurn from Subroutine (von Subroutine zurückspringen)
Trifft das Programm auf den Befehl RTS, springt der Prozessor auf die Speicherstelle zurück, von der aus die Unterroutine aufgerufen wurde (entspricht hier dem Befehl RETURN in BASIC).

Branch if MInus (verzweige bei Minuswert)
Dieser Befehl ist eine von mehreren bedingten Verzweigungsmöglichkeiten im Maschinencode. (Der BASIC-Befehl IF...THEN GOTO ist ebenfalls ein bedingter Sprung.) Befindet sich ein negativer Wert im Akkumulator, dann verzweigt die Programmausführung zu der angegebenen Adresse. BPL veranlaßt den Sprung bei einem positiven Wert.

LoaD X register (Lade das X-Register)
Das X-Register ist ein weiterer interner Speicher. Es kann zwar keine Rechenvorgänge ausführen, wird aber für die „indexierte Adressierung" eingesetzt. LDX lädt einen Wert in das Register X, während STX (STore X) diesen Wert wieder in den Speicher zurückschreibt.

INcrement X (Erhöhe X um 1)
Addiert man eine 1 auf den Wert des Registers X (DEX – DEcrement X – subtrahiert 1 vom Wert des Registers X) und verwendet die indexierte Adressierung, kann man Schritt für Schritt durch eine Anzahl Speicherstellen hindurchgehen.

Motorola 68000

Seit Motorola im Jahre 1982 mit den Microprozessoren der 68000er Serie den Nachfolger der vielgerühmten 8-Bit-CPU 6809 herausbrachte, dominiert dieser Prozessor auf einem wesentlichen Teil des Marktes für 16- und 32-Bit-Prozessoren.

Microprozessoren

6502
Wie der Z80 von Zilog ist auch der von MOS Technology entwickelte 6502 einer der Hauptprozessoren der Microcomputerindustrie. Er besitzt einen 16-Bit-Adreßbus und einen 8-Bit-Datenbus. Größte Eigenheit des Prozessors ist seine Registeranordnung. Die gesamte Zeropage (der erste Block des RAM-Bereiches) kann als Allzweckregister verwandt werden.

68000
Motorola verbesserte zwar den 6800 zum 6809 Chip, konnte sich aber nicht mehr rechtzeitig einen größeren Anteil des 8-Bit-Marktes sichern. Im nachhinein gesehen war das ein Vorteil, da die Firma sich dadurch auf die Entwicklung des 68000, einen 16/32-Bit-Prozessor, konzentrieren konnte. Der 68000 läßt sich in Verbindung mit vielen Zusatzchips betreiben.

Z80
Theoretisch verfügt der Z80 von Zilog über größere Kapazitäten als der MOS 6502. Er verwendet ähnliche Adreß- und Datenbusstrukturen, hat jedoch wesentlich mehr Register (12 8-Bit-Allzweckregister, zwei 16-Bit-Indexregister) und einen weitaus größeren Befehlssatz. Sein vielleicht größter Vorteil gegenüber dem 6502 ist die Unterstützung des Betriebssystems CP/M.

Der 68000 hat starke Ähnlichkeit mit dem MC6800 Prozessor von Motorola, der als intelligenter Steuerchip für Peripheriegeräte weit verbreitet ist. Der weitaus höher entwickelte 68000 läßt sich aus diesem Grund leicht mit Peripheriegeräten verbinden, wobei eine breite Palette einsatzfähiger Hardware bereits zur Verfügung steht. Ein- und Ausgabeplatinen mit dem 6821 PIA Chip (Peripheral Interface Adaptor) lassen sich dabei ebenso leicht anschließen wie Bildschirmgeräte mit 6845 CRTC Steuerchips (Cathode Ray Tube Controller), Taktgeber mit dem programmierbaren Zeitmodul des 6840 oder Diskettencontroller.

Der 68000 ist für die Entwicklung von Computern besonders wegen der großen Anzahl seiner Daten- und Adreßleitungen interessant. Beide Busnetze haben voneinander unabhängige Leitungen, wobei jedes einzelne Bit über einen eigenen Kontaktpin verfügt. Bei dem 8086, 8088 und Z8000 laufen die Kontakte im Multiplexverfahren zusammen, d. h., zwei Busse teilen sich einen Kontaktpin, wobei die ausgegebenen Signale von den angesprochenen Peripheriegeräten wieder getrennt und decodiert werden müssen.

Der 68000 kann daher so schnell arbeiten, wie seine Systemumgebung es erlaubt. Beim Einsatz entsprechend schneller RAM-Chips lassen sich Warteperioden auf ein Minimum reduzieren. Der schnellste Prozessor dieser Serie ist der 68000L12, der in vielen Funktionsbereichen bis 16 Megahertz getaktet werden kann.

Sinclair setzte für seinen QL den 68008 (den Nachfolger des 68000) ein. Intern arbeitet der Prozessor auf die gleiche Weise wie die anderen Prozessoren der Serie. Da er jedoch mit einem Acht-Bit- statt mit einem 16-Bit-Datenbus ausgerüstet ist, ist er mit bestehenden Acht-Bit-Systemen kompatibel. Dieser Chip wird im Format von 40 Kontaktpins geliefert.

Motorola wird bald einen Microprozessor mit noch größeren Verarbeitungsgeschwindigkeiten vorstellen. Es handelt sich um den 68020, einen 32-Bit-Prozessor mit 96 Kontaktpins, dessen Entwicklung jedoch noch nicht abgeschlossen ist. Ein speziell für mathematische Fließkommaarithmetik ausgelegter Prozessor ist ebenfalls in Vorbereitung.

Zusätzliche Funktionen

Weitere Chips der 68000-Serie bieten Ein- und Ausgabefunktionen, die denen früherer Modelle strukturell ähnlich sind, jedoch weitaus schneller arbeiten und über zusätzliche Funktionen verfügen.

Im Vergleich zu anderen 16-Bit-Prozessoren bietet der 68000 aufgrund seiner symmetrisch angelegten Adreß- und Datenregister und seines reichhaltigen Befehlssatzes eine ganze Reihe von Vorteilen. Vollkommen ist er jedoch nicht. Obwohl Adreß- und Datenregister die gleiche Breite (32 Bits) haben und in vielen Funktionen mit identischen Befehlen arbeiten, können sie nur separat eingesetzt werden. Daten müssen daher zur Bearbeitung oft von dem Adreßregister erst in ein Datenregister übertragen und nach der Bearbeitung wieder an das Adreßregister zurückgesandt werden.

Generell finden die schnellen und effektiven Prozessoren der 68000-Serie mehr und mehr Verbreitung. Sie sind in den Lisa und Macintosh von Apple ebenso eingebaut wie in den QL von Sinclair, und auch viele Mehrplatzcomputer, die auf dem Markt weniger bekannt sind, verwenden diesen Chip. Fähigkeiten, die vor einigen Jahren noch viel Geld gekostet hätten, sind nun zu vernünftigen Preisen verfügbar. Für die nächste Computergeneration scheint die 68000-Serie den gleichen Anklang zu finden, wie der Z80 und der 6502 sie heute besitzen.

Interne Funktionsabläufe

Mit der Maschinensprache lassen sich alle Funktionen des Microprozessors direkt steuern. Sie ist der Schlüssel zur wahren Beherrschung eines Computers. Hier beginnt ein umfassender Kurs, in dem wir schrittweise die Maschinensprache der Microprozessoren 6502 und Z80 erklären.

Der Maschinencode kann folgende Formen annehmen:

```
INSTK: SBC $D9FA,X;Outport flag value
```

oder so:

```
DE23 FD FA D9
```

oder aber:

```
11011110 00100011 11111101 11111010
11011001
```

Manchmal auch so:

```
1240 LET ACC=ACC-FLAG(X)
```

oder so:

```
PERFORM FLAG-ADJUST THROUGH LOOP1
```

Es gibt viele Möglichkeiten, dasselbe auszudrücken. Alle haben direkten Einfluß auf die Funktionen des Computers und werden deshalb Maschinensprache oder Maschinencode genannt. Auf der anderen Seite ist es der Maschine gleich, welche Form der Code annimmt – für sie ist er immer eine Folge von elektronischen Impulsen.

Lösungswege

Mit dem Ausdruck „Maschinencode" ist meistens die Assemblersprache gemeint. Das erste Beispiel stellt eine Befehlszeile für den 6502 dar. Alle anderen Beispiele wurden angeführt, um zu zeigen, daß es keinen Standard der Maschinensprache gibt, sondern nur eine Anzahl unterschiedlicher Methoden, die Folgen von elektrischen Impulsen mehr oder weniger leicht lesbar darzustellen. Wir wollen daher den Maschinencode (oder Assembler – wir gehen im Augenblick noch nicht auf Unterschiede ein) zunächst nur als eine normale Programmiersprache ansehen. Wichtig ist jedoch, daß dabei die Programmierung immer im Vordergrund steht: Ganz gleich, ob Sie in IBM-Assembler oder in Atari-BASIC schreiben, bevor Sie auch nur ein Zeichen eingeben, müssen Sie eine Vorstellung des Lösungsweges haben.

Wozu braucht man den Maschinencode, und was bedeutet der Name? Zunächst: Der Name wurde gewählt, weil dieser Befehlssatz mit den grundlegenden Funktionen der Microprozessoren übereinstimmt. Der Maschinencode wird eingesetzt, wenn die Abläufe im Microprozessor Schritt für Schritt überwacht werden sollen, statt die Steuerung an den Interpreter einer Programmsprache zu übergeben, der sich nur indirekt kontrollieren läßt.

Generell wird im Maschinencode programmiert, wenn höhere Ablaufgeschwindigkeiten notwendig sind. Bei der direkten Arbeit mit dem Prozessor ersparen Sie sich den relativ langsamen Vorgang der Programmübersetzung. Genauer gesagt: Sie sparen Programmablaufzeit, aber die Eingabe, das Testen, die Fehlersuche, Änderungen und die Dokumentation nehmen im Maschinencode weitaus mehr Zeit in Anspruch als bei einer Hochsprache. Die Unbeweglichkeit und Komplexität der Maschinensprache hat nicht umsonst die Entwicklung von Hochsprachen wie COBOL und BASIC beeinflußt.

Wenn der Befehlssatz des Maschinencodes und die Funktionen des Microprozessors übereinstimmen, wie sehen dann diese Abläufe aus und wie arbeitet der Prozessor? In einfachen Worten beschrieben ist die Zentraleinheit eines Computers ein Schalter, der den Datenaustausch zwischen einzelnen Teilen des Systems steuert. Die Komponenten sind der Arbeitsspeicher, die Einheit für Arithmetik und Logik und die Ein- und Ausgabegeräte. Wenn Sie auf der Tastatur ein Zeichen eingeben wollen und eine Taste drücken, erzeugen Sie auf Maschinenebene eine Folge von elektrischen Impulsen. Die CPU leitet diese Folge in einen bestimmten Teil des Speichers, ruft aus einem anderen Speicherbereich eine entsprechende Impulsfolge ab und sendet diese zum Bildschirm, wo wiederum elektronische Impulse das Zeichen darstellen. Der Vorgang mag wie die Bedienung einer Schreibmaschine ausse-

hen. Bei ihr besteht zwischen der Taste und dem Druck des Zeichens eine mechanische Verbindung, während in einem Computer diese Verbindung von der CPU hergestellt wird, die die entsprechenden Impulsfolgen von einem Geräteteil zum anderen leitet. In manchen Fällen erscheint dabei noch nicht einmal ein Zeichen auf dem Bildschirm: Der Tastendruck kann zum Beispiel einen Asteroiden zerstören, ein Programm speichern oder eine Datei löschen, – die ausgeführte Funktion hängt davon ab, wohin die CPU die Impulsfolgen leitet.

In diesem vereinfachten Modell stellt die CPU das Systemherz dar, das alle Informationen (oder elektrische Impulse) passieren müssen, um von einem Teil des Systems in einen anderen gelangen zu können. In Wirklichkeit sieht die Arbeitsweise der CPU und des Systems jedoch wesentlich komplizierter aus. Stellen Sie sich die Zentraleinheit am besten als eine Hauptsteuerung vor, die verschieden gestaffelte Hierarchien von Schaltern betätigen kann, mit denen sie den Fluß der Information indirekt leitet.

Impulsfolgen

Die Schaltoperationen der CPU lassen sich in bestimmte Bereiche aufteilen, die sich deutlich voneinander unterscheiden. Es gibt arithmetische, logische und speicherbezogene Abläufe wie auch Steuervorgänge. All diese Aktivitäten werden von Impulsfolgen ausgelöst, die auf unterschiedlichen Wegen durch das System geleitet wurden, wobei die CPU selbst die unterschiedlichsten Vorgänge immer auf die gleiche Art zu steuern scheint.

Die Steuerung arithmetischer Abläufe ist mit Abstand die wichtigste Funktion der Maschine. Die CPU kann zwei Zahlen addieren oder subtrahieren, wobei die Subtraktion eine Zahl in ihren negativen Wert verkehrt und dann zu der zweiten Zahl addiert. 7+5=12 bedeutet:

(plus 7) addiert mit (plus 5) ergibt (plus 12).

7–5=2 bedeutet:

(plus 7) addiert mit (minus 5) ergibt (plus 2).

Multiplikation und Division lassen sich als wiederholte Addition und Subtraktion ausdrücken und können dadurch von der Zentraleinheit ebenfalls ausgeführt werden. Mit den vier Grundrechenarten aber kann die CPU jeden mathematischen Vorgang bewältigen, wobei ihr gesamtes mathematisches Potential nur auf der einfachen Fähigkeit beruht, zwei Zahlen addieren zu können.

Logische Vorgänge wollen wir im Augenblick als die Fähigkeit beschreiben, zwei Zahlen miteinander vergleichen zu können. Damit sind aber nicht nur einfache Größenvergleiche gemeint, sondern auch die Untersuchung der Zahlenmuster einzelner Stellen. Die Tatsache, daß sieben größer als fünf ist, läßt sich leicht feststellen, wenn von sieben fünf abgezogen wird und das Ergebnis positiv ist. Die CPU verfügt über diese Fähigkeit, kann aber auch z. B. die Zahlen 189 und 102 miteinander vergleichen und feststellen, daß beide Zahlen in den Hundertern die gleiche Ziffer haben.

Die CPU kann grundsätzlich zwei Speichervorgänge ausführen: Sie kann die Informationen einer Speicherstelle in ihren internen Speicher kopieren und von diesem an eine andere Speicherstelle setzen. Werden beide Vorgänge nacheinander ausgeführt, können damit Informationen aus jedem Speicherbereich in jeden anderen Speicherbereich kopiert werden. Mit diesen beiden Fähigkeiten läßt sich das gesamte Speichermanagement vollständig bewältigen.

Steuervorgänge sind Entscheidungen der CPU, in welcher Reihenfolge sie die oben beschriebenen Vorgänge ausführt. Die CPU kann mathematische Berechnungen ausführen, Zahlen miteinander vergleichen, Informationen im Speicher bewegen und Entscheidungen über die internen Funktionsabläufe fällen. Beherrscht eine Zentraleinheit diese vier Abläufe dann kann sie alle nur denkbaren Aufgaben ausführen, die für Computer geeignet sind. Die Abläufe müssen dafür nur in die richtige Reihenfolge gestellt werden, wobei die Reihenfolge durch das Programm definiert wird, das für die Lösung einer speziellen Aufgabe analysiert wurde.

Diese vier Funktionsarten reichen für die Beschreibung eines Computerkonzeptes aus.

Funktionsblöcke

Sehen wir uns einmal die Funktionsblöcke eines Programms in BASIC an. Welche Teile kann man darin unterscheiden? Jedes Programm hat Variablen, die nichts anderes sind als die Namen der Speicherbereiche, in denen bestimmte Informationen abgelegt sind. Die meisten Programme führen mit diesen Variablen mathematische Operationen durch. Nach der Berechnung vergleicht ein Programm in vielen Fällen zwei Informationen und entscheidet sich auf der Grundlage des Ergebnisses, ob es eine bestimmte Gruppe von Befehlen ausführt. Informationen erhält ein Programm üblicherweise über die Tastatur.

Abgesehen von der Ein- und Ausgabe enthält diese Beschreibung nicht mehr als die vier Grundfunktionen der CPU – nur in anderen Worten beschrieben. Und wenn Sie zunächst akzeptieren, daß die CPU Ein- und Ausgabegeräte nur als besondere Speicherbereiche ansieht, dann ist diese Beschreibung der Funktionsabläufe sogar komplett. Die Ausführung eines Programms läßt sich als Steuerung eines Informationsflusses beschreiben.

Dieser Beitrag über die internen Funktionsabläufe wird im nächsten Heft fortgesetzt.

Abläufe in der CPU

Diese Folge über den Maschinencode beschäftigt sich mit den Funktionen der CPU. Ein Listing für den Acorn B und eins für den Commodore 64 werden Ihnen dabei helfen, die Vorgänge in der Zentraleinheit leichter zu verstehen.

Da die Zentraleinheit eines Computers eine wichtige Funktion erfüllt, wollen wir untersuchen, wie sie arbeitet und welche Funktionen sie ausführt.

Stellen Sie sich eine einfache elektrische Schaltung vor, die aus einer Batterie, einem Schalter und einer Glühbirne besteht. Wird der Schalter geschlossen, geht das Licht an und leuchtet so lange, bis die Batterie leer ist oder der Schalter wieder geöffnet wird. Der Zustand der Glühbirne – AN oder AUS – ist dabei eine Information, während die Schaltung diese Information speichert. Nehmen wir an, der Schalter ist neben dem Eingang einer Fabrik installiert und die Glühbirne befindet sich im Büro des Managers. Wenn der erste Angestellte an das Fabriktor kommt, betätigt er den Schalter. Der Chef in seinem Büro sieht, daß die Glühbirne leuchtet, und weiß, daß jemand zur Arbeit erschienen ist. Der Manager braucht nicht einmal in seinem Büro zu sein, wenn das Licht angeht. Er kann anhand der Glühbirnenschaltung jederzeit erkennen, ob jemand angekommen ist, sofern diese Informationen gespeichert wurden.

Auf fast die gleiche Weise speichert ein Computer die Daten: Alle Informationen werden auf den Zustand elektrischer Schaltungen zurückgeführt. Der Vorgang ist natürlich etwas komplizierter, und daher werden wir unser Informationssystem noch ein wenig verfeinern.

Informationssprache

Bei der Verwendung von vier Stromkreisen (vier Schalter in einer Reihe neben der Eingangstür und die Glühbirnen in der entsprechenden Reihenfolge im Büro des Managers) leuchtet beim Betätigen des linken Schalters auch die linke Glühbirne auf etc. Jedem Angestellten wird mitgeteilt, die Schalter nur auf eine individuell bestimmte Weise zu betätigen, bei der z. B. Katrin den ersten und zweiten Schalter schließt, den dritten und vierten aber öffnet, Richard den vierten schließt und alle anderen Schalter öffnet und Klaus den ersten und dritten schließt, den zweiten und vierten öffnet und so weiter für alle Angestellten. Die optischen Signale im Büro des Managers zeigen jetzt genau an, welcher Angestellte angekommen ist.

Wenn wir die Position AUS als 0 bezeichnen und die Position AN als 1, dann hat Katrin die Schalter auf 1100 (die ersten beiden Schalter AN und den dritten und vierten AUS) gestellt, Richard auf 0001 (den vierten Schalter auf AN und alle anderen auf AUS) und Klaus auf 1010 (den ersten und dritten AN und den zweiten und vierten AUS). Interpretiert der Manager eine eingeschaltete Glühbirne ebenfalls als 1 und eine ausgeschaltete als 0, dann „sprechen" sowohl er als auch die Angestellten die gleiche Informationssprache. „0001" bedeutet für alle dabei eindeutig „Richard".

Wie viele eindeutige Kombinationen lassen sich mit den Schaltern herstellen? Es gibt vier Schalter, die in je zwei Stellungen stehen können, also 2×2×2×2=16 unterschiedliche Kombinationsmöglichkeiten:

0000,0001,0010,0011,0100,0101,0110,0111,
1000,1001,1010,1011,1100,1101,1110,1111

So läßt sich das konkrete Modell der Glühbirnen auf abstrakte Folgen von Einsen und Nullen übertragen. Wenn man in der Abstraktion nur ein wenig weiter geht, lassen sich diese Folgen in Zahlen umwandeln.

Stellen Sie sich vor, Sie zählen und schreiben dabei die Zahlen auf. Von Null bis Neun geht es schnell, da jede dieser Zahlen einen eigenen Namen hat und von einem eigenen Symbol dargestellt wird. Was passiert aber, wenn Sie Zahlen über Neun aufschreiben wollen? Die Zahl hat zwar einen eigenen Namen – Zehn –, läßt sich aber nicht mit einem einzigen Symbol darstellen. Aus diesem Grunde setzen Sie sie aus den bereits geschriebenen Symbolen zusammen: 10,11,12 etc. bis 99. Jetzt sind auch alle zweistelligen Kombinationen erschöpft und die nächste – dritte – Stelle wird eröffnet (100). Dieser Vorgang erscheint trivial, doch denken Sie daran, wie viele Schwierigkeiten gerade die Kästchen mit den Hundertern, Zehnern und Einern einem Schulanfänger bereiten können.

Unser Zahlensystem beinhaltet 10 Ziffern (0,1,2,3 . . .9), aus denen wir alle Zahlen zusam-

mensetzen. Wie aber funktioniert das Zählen, wenn man nur über zwei Symbole verfügen kann: 0 und 1? Bis eins können wir leicht zählen, wie aber läßt sich die nächste Zahl darstellen? Da keine weiteren Symbole zur Verfügung stehen, müssen die vorhandenen wie bei dem Zehnersystem zu Kombinationen zusammengesetzt werden. Die auf eins folgende Zahl ist also 10. Die nächste Zahl heißt Drei und wird in Form von 11 geschrieben. Da jetzt alle zweistelligen Kombinationsmöglichkeiten erschöpft sind, muß die Zahl Vier dreistellig (= 100) sein, die Fünf (101), die Sechs (110) und die Sieben (111) ebenfalls. Wir zählen im Dezimalsystem

Diese zwei kurzen Programme für den Acorn B und den Commodore 64 bringen den vorhandenen Zeichensatz auf den Bildschirm und zeigen die Unterschiede zwischen BASIC und Maschinencode.

Acorn B

```
100 REM*********************************
**********************BBC****************
149 REM*****************************
150 REM*      BBC M/C CODE DEMO     *
151 REM*****************************
200 MODE 4:*TV 254
300 GOSUB 30000
700 FOR P=1920 TO 6079
800 K=K+1:IF K>2679 THEN K=1920
900 ?(HIMEM+P)= (K+47232)
1000 NEXT P
1100 PRINT TAB(13);"THAT WAS BASIC"
1200 INPUT" HIT RETURN FOR MACHINE CODE
VERSION ",A$:CLS
1300 FOR L=0 TO 15:FOR B=0 TO 255 STEP L
1400 ?(SA)=LS:?(SA+1)=HS
1500 ?(FA)=LF:?(FA+1)=HF
1600 DUMMY=USR(PSTRT)
1700 VDU 30
1800 NEXT B,LP
1900 STOP
30000 REM******MC LOADER S/R****
30010 K=1919:PSTRT=PAGE+8:VSTR=HIMEM+192
0
30020 HS=INT(VSTR/256):LS=VSTR-256*HS:LF
=LS+56:HF=HS+2:SA=114:FA=116
30100 DATA 50,169,32,197,112,48,4,240,2,
133,112,165,112,32,227,255
30110 DATA 230,114,208,2,230,115,165,116
,197,114,208,7,165,117
30120 DATA 197,115,208,1,96,230,112,169,
128,197,112,208,224
30130 DATA 169,32,133,112,208,218,96,96,
96
30150 READ ZZ
30160 FOR BY=PSTRT TO PSTRT+ZZ
30170 READ MC:?(BY)=MC
30180 NEXT BY
30200 RETURN
30299 REM*****************************
30300 REM*   SAVE BEFORE RUNNING !! *
30301 REM*****************************
30399 REM*****************************
30400 REM*DO NOT LIST LINE 100 AFTER*
30401 REM*     RUNNING PROGRAM !!    *
30402 REM*****************************
```

Commodore 64

```
99 REM ************************
100 REM*COMMODORE M/C CODE DEMO *
101 REM************************
200 PRINT CHR$(147)  :REM CLEAR SCREEN
300 PRINT "    THIS WON'T TAKE LONG"
400 GOSUB 60000
500 PRINT CHR$(147);CHR$(5) :REM CLS AND
 WHITE
600 CC=0
700 FOR P=SM TO FM
800 POKE P,CC:POKE P+OF,CL
900 CC=CC+1:IF CC>255 THEN CC=0
1000 NEXT P
1100 PRINT TAB(13);"THAT WAS BASIC"
1200 INPUT" HIT RETURN FOR MACHINE CODE
VERSION ";A$
1300 FOR LP=1 TO 9:FOR B=0 TO 255 STEP L
P
1400 POKE SA,LS:POKE SA+1,HS
1500 POKE FA,LF:POKE FA+1,HF
1600 POKE BA,B:POKE CH,0
1700 SYS AA
1800 NEXT B,LP
1900 STOP
60000 REM******MC LOADER S/R****
60010 SM=256*PEEK(648):OF=55296-SM:FM=SM
+999:BD=53280:SC=BD+1:CS=8:CB=6:CL=0
60020 POKE BD,CB:POKE SC,CS
60030 LS=0:HS=PEEK(648):LF=232:HF=HS+3:S
A=251:FA=253:BA=250:CH=2
60100 DATA 850,885,169,0,170,165,250,133
,2,165,2,129,251
60110 DATA 230,251,208,2,230,252,165,253
,197,251,208,7,165,254
60120 DATA 197,252,208,1,96,230,2,208,22
9,240,223
60150 READ AA,ZZ
60160 FOR BY=AA TO ZZ
60170 READ MC:POKE BY,MC
60180 NEXT BY
60200 RETURN
60299 REM*****************************
60300 REM* SAVE THIS BEFORE RUNNING IT*
60301 REM*****************************
```

(Null, Eins, Zwei etc.), schreiben die Zahlen im binären System aber so: 0, 1, 10, 11, 100, 101, ...

Nach genau der gleichen Methode, in der eine Dezimalzahl wie 152 eigentlich (1×100) + (5×10) + (2×1) darstellt, bedeutet die Binärzahl 101: (1×4) + (0×2) + (1×1). Statt Stellen mit Hundertern, Zehnern und Einern für unsere Zahlen zu verwenden, bedeuten die Stellen im Binärsystem Vierer, Zweier und Einer. Bewegt man sich in einer Dezimalzahl von rechts nach links, erhöht sich der Wert jeder Stelle fortlaufend um zehn. Bei einer Binärzahl erhöhen die Stellen sich jeweils um zwei.

Erinnern wir uns noch einmal kurz an unsere Analogie der Fabrikangestellten, die Schalter nach einem individuellen Muster zu betätigen. Die Schalterstellungen lassen sich dabei als vierstellige Binärzahlen ansehen. Das Signal von Katrin ist dann binär 1100 oder dezimal 12. Richards Signal ist binär 0001 (dezimal 1) und das Signal von Klaus binär 1010 (dezimal 10).

Schaltmuster

Wenn der Manager jetzt die Schaltmuster der Glühbirnen ansieht, kann er sie als binäre Zahlen verstehen, in ihre dezimale Entsprechung umwandeln und nachsehen, welchem Namen die Zahl entspricht.

Unser Modell zeichnet ein einfaches Bild, wie in einem Computer Informationen dargestellt werden. Für den Computer existieren nur Folgen von elektrischen Impulsen (Lichter, die den Zustand AN oder AUS darstellen); für uns Menschen ist es aber einfacher, diese Impulsfolgen als Binärzahlen anzusehen. Wenn Sie jetzt daran denken, daß der Code 1010 dem Namen „Klaus" entspricht, dann verstehen Sie, wie Zahlen in der Maschinensprache eingesetzt werden können. In unserer nächsten Folge werden wir untersuchen, wie ein Heimcomputer Informationen durch Binärzahlen darstellt.

Speicheraufbau

In dieser Folge über die Maschinencode-Programmierung zeigen wir, wie ein Computer seine Daten speichert, sein „Gedächtnis" organisiert und auf welche Weise die CPU einzelne Bytes mit Binärzahlen adressiert.

In der ersten Folge haben wir erklärt, wie ein Computer Informationen durch elektrische Impulse speichern kann. Bei dem Beispiel wurde deutlich, wie die vier Schalter und Glühbirnen mit ihren 16 unterschiedlichen Schaltmöglichkeiten die Zahlen 0 bis 15 darstellen können. Mit acht Schaltern und Glühbirnen ergeben sich jedoch 256 individuelle Schaltmuster ($2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 256$) für die Zahlen von 0 bis 255.

Der Arbeitsspeicher eines Heimcomputers ist aus Blöcken aufgebaut, die „Bytes" genannt werden und aus je acht Schaltern bestehen. Im allgemeinen kann die CPU nur ein Byte zur Zeit bearbeiten und damit auch nur Zahlen zwischen 0 und 255 addieren, vergleichen oder speichern. Auf den ersten Blick scheint das die Arbeit der CPU stark einzuschränken. Bedenken Sie aber, daß Sie bei der Addition zweier Zahlen (z. B. 63951 + 48770 = ?) auch nur die einzelnen Stellen nacheinander berechnen. Auf genau die gleiche Weise führt die CPU mathematische Aufgaben mit großen Zahlen Byte für Byte durch.

Mit seinen acht Schaltern kann ein Byte eine achtstellige Binärzahl speichern. Jede einzelne Stelle dieser Binärzahl wird „Bit" genannt und stellt die kleinstmögliche Informationseinheit dar. Ein Bit ist entweder AN oder AUS, und eine binäre Stelle ist entweder 1 oder 0.

MSB – Most Significant Bit

Da die einzelnen Bits eines Bytes oft einzeln angesprochen werden, bezeichnet man sie von links nach rechts mit den Zahlen 0 bis 7. Enthält z. B. ein Byte die Binärzahl 00000001, dann heißt es, Bit0 ist 1, oder Bit0 ist AN, oder Bit0 ist gesetzt, und alle anderen Bits sind 0 oder AUS oder CLEAR. In der Binärzahl 01001000 sind daher Bit3 und Bit6 gesetzt, Bit4 ist AUS, Bit7 ist 0, Bit0 ist nicht gesetzt etc. In einem Byte wird Bit0 auch das „niederwertigste Bit" (LSB – Least Significant Bit) genannt, und Bit7 das „höchstwertige Bit" (MSB – Most Significant Bit).

Den Speicher eines Computers kann man sich auch als ein großes Blatt Rechenpapier vorstellen, das acht Kästchen breit und Tausende von Kästchen lang ist. Jede Reihe von acht Kästchen ist ein Byte und jedes Kästchen ist ein Bit innerhalb eines Bytes. Da ein Speicher völlig nutzlos wäre, wenn man die darin enthaltenen Informationen nicht wiederfinden könnte, besitzt jedes Byte eine eigene Bezeichnung: die Adresse. Diese wird nun nicht extra in jeder Kästchenreihe (oder in jedem Byte) geschrieben, sondern ergibt sich aus der Position des Bytes vom Anfang des Speichers an gezählt. Das erste Byte im Speicher hat die Adresse 0, das nächste Byte die Adresse 1, das nächste die Adresse 2 und so weiter. Wenn Sie etwas in das Byte43 schreiben möchten, fangen Sie am Anfang des Speichers (bei Byte0) an, und zählen die Bytes, bis Sie Byte43 erreicht haben.

Offset und Seitenoffset

Die Adresse 43 des Byte43 wird dabei nur von der Position bestimmt – Sie haben von Byte0 aus 43 Bytes abgezählt und erreichten somit Byte43. In den Chips im Inneren des Computers sind die Bytes des Speichers als winzige Blöcke mit je acht Transistorschaltungen (pro Bit eine Schaltung und pro Byte acht Schaltungen) eingeätzt, deren Konstruktion, abgesehen von ihrer Position, identisch ist.

Stellen wir uns den Speicher eines Computers nochmals als Papierstreifen mit einer Breite von acht und einer Länge von Tausenden von Kästchen vor. Dieser Streifen ließe sich an jedem hundertsten Byte (d. h., an der Grenze zwischen dem Byte99 und dem Byte100, zwischen Byte199 und Byte200, Byte299 und Byte300 etc.) zerschneiden. Jeder dieser Teilstreifen wäre jetzt eine Seite mit je 100 Bytes. Seite0 (Page0) fängt bei Byte0 an und geht bis Byte99, Seite1 beginnt bei Byte100 und reicht bis Byte199, und Seite2 reicht von Byte200 bis Byte299 etc. Wenn wir jetzt ein bestimmtes Byte suchen, z. B. das Byte3518, brauchen wir nicht vom Anfang des Speichers an 3518 Bytes abzuzählen, sondern können aus der Adresse des Bytes ersehen, daß es auf der Seite 35 liegen muß. Wir brauchen daher vom Anfang des Speichers nur 35 Seiten abzuzählen und dann vom Anfang dieser Seite 18 Bytes, um Byte3518 zu erreichen.

Eine Aufteilung des Speichers in Seiten ist außerordentlich praktisch, da man jedes Byte leicht finden kann, wenn seine Adresse in zwei Teile zerlegt wird. Die Ziffern von der Hunderterposition aus nach links geben dabei die Sei-

tennummer an und die Ziffern von der Zehnerposition aus nach rechts die exakte Position des Bytes auf dieser Seite. In unserem Beispiel teilen wir die Adresse des Bytes 3518 in zwei Zahlen auf: Seitennummer 35 und Byte Nummer 18 auf einer Seite. Die Zahl 18 wird dabei auch „Offset" oder „Seitenoffset" genannt und bezeichnet den Abstand oder die Distanz einer Adresse zu einer Basisadresse (in diesem Fall ist die Basisadresse Byte3500).

Ein Computer zählt jedoch nicht dezimal, sondern binär. Mit dem Seitensystem muß sich dabei eine bestimmte Seite und ein Offset ebenso leicht wie im Dezimalsystem über die Aufteilung der Adresse finden lassen. Nun wird die Dezimaladresse 99 binär von der Zahl 01100011 dargestellt und dezimal 100 von der Binärzahl 01100100, dezimal 199 = binär 11000111 und dezimal 200 = binär 11001000.

Die Zahl 100 bietet sich im Dezimalsystem als Seitengröße an, da sie ein Vielfaches von 10 ist und sich von anderen Zahlen deutlich unterscheidet. Wenn wir im Binärsystem zählen, müssen wir also eine Einheit wählen, die für dieses System Bedeutung hat. Microcomputer verwenden daher Seitengrößen von 256 Bytes, so daß Seite0 bei Byte0 anfängt und bei Byte255 aufhört, Seite1 bei Byte256 beginnt und bis Byte511 reicht etc. Der Grund für diese Einteilung läßt sich leicht aus der binären Darstellung dieser Zahlen ersehen:

Seite0: Byte00000000 bis Byte11111111
Seite1: Byte100000000 bis Byte111111111

Wie Sie sehen, lassen sich die Zahlen 0 bis 255 mit acht Bits darstellen, die nächste Zahl – 256 – benötigt neun Bits, und mit neun Bits können wir bis 511 zählen. Die nächsthöhere Zahl – 512 – benötigt zehn Bits, mit denen wir bis 1023 zählen können usw. Wenn die Seitengröße also 256 ist und wir im Binärsystem zählen, dann sind die acht Bits auf der rechten Seite der Zahl der Seitenoffset, während sich die Seitennummer aus den Bits ergibt, die von Bit8 an auf der linken Seite der Zahl stehen.

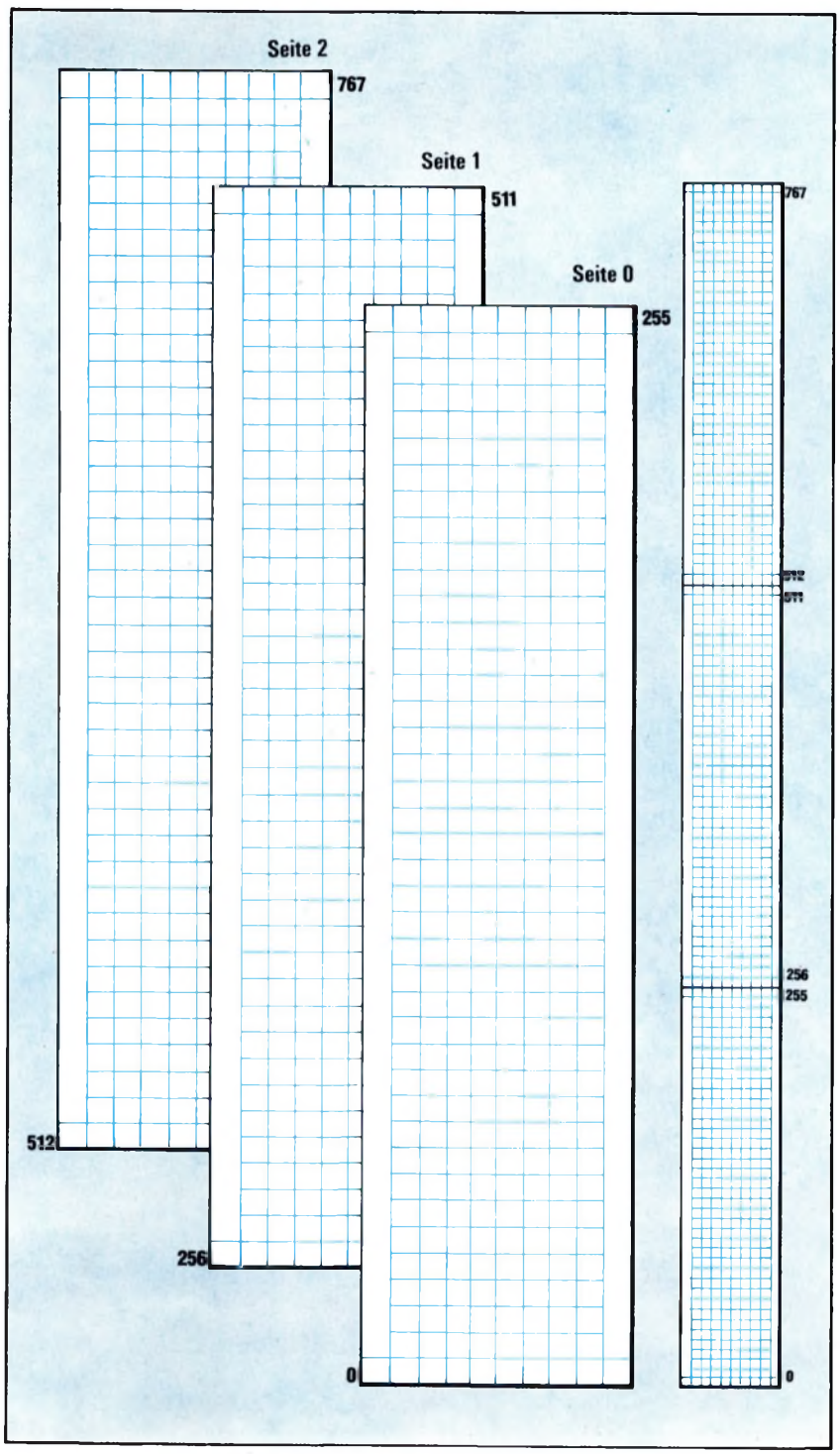
Speicheradressen aus zwei Bytes

Diese Einteilung mag ein wenig verwirren, da wir früher erwähnt hatten, daß die CPU nur einzelne Bytes verarbeiten kann. Wenn ein Byte nur acht Bits enthält, warum dann Zahlen mit neun oder zehn Bits verwenden? Die Antwort ist einfach: Alle Speicheradressen bestehen aus zwei Bytes, wobei die CPU die beiden Bytes nacheinander bearbeitet. Wenn wir die Seitengrenzen als Zwei-Byte-Zahlen darstellen, wird das gesamte System deutlich:

Seite	0	beginnt bei	00000000 00000000
		endet bei	00000000 11111111
Seite	1	beginnt bei	00000001 00000000
		endet bei	00000001 11111111
Seite	10	beginnt bei	00000010 00000000
		endet bei	00000010 11111111
Seite	11	beginnt bei	00000011 00000000
		endet bei	00000011 11111111

und so weiter.

Aus diesen Beispielen läßt sich leicht ersehen, wie die CPU ein Byte im Speicher über eine Zwei-Byte-Adresse anspricht. Das erste oder links stehende Byte ergibt die Seitennummer und das zweite oder rechte Byte den Offset, die Position der Daten.



Die Seitenadressierung unterteilt den gesamten Speicher in Blöcke oder Seiten von je 256 Bytes Länge. Alle Adressen werden als Zwei-Byte Zahlen dargestellt: Ein Byte gibt die Seitennummer an, während das andere Byte den Abstand vom Anfang dieser Seite (Offset) enthält.

Zahlen-umwandlung

Diese drei Programme für den Commodore 64, den Acorn B und den Spectrum wandeln dezimale Eingaben in die binären Äquivalente um.

Commodore 64

```
10 REM********COMMODORE**********
40 S$="=          ":X$="0123456789ABCDEF"
50 REM   S$ CONTAINS 9 SPACES
60 PRINT CHR$(147)    :REM CLEAR SCREEN
70 PRINT "      TO DISPLAY DECIMAL NUMBER
S"
80 PRINT "     AND THEIR BINARY EQUIVALEN
TS"
90 PRINT:PRINT "    *******ENTER 0 TO QUI
T********":PRINT
100 FOR K=1 TO 1
110 FOR L=1 TO 1
120 INPUT"TYPE ANY POSITIVE WHOLE NUMBER
 ";A$
130 NU=VAL(A$)
140 IF NU=0 THEN PRINT "PROGRAM EXIT":ST
OP
150 IF INT(NU)<>ABS(NU) THEN L=0
160 IF NU>65535 THEN PRINT NU;" IS TOO B
IG":L=0
170 NEXT L
200 NM=NU:H$="":GOSUB 2000
210 PRINT NU;TAB(5);N$;
220 IF RIGHT$(A$,1)="+" THEN GOSUB 4000
230 PRINT H$:PRINT:PRINT
240 K=0:NEXT K
250 END
300 END
1000 REM******BINARY BYTE S/R*******
1010 B$=""
1020 FOR D=8 TO 1 STEP-1
1030 N1=INT(N/2)
1040 R=N-2*N1
1050 B$=MID$(STR$(R),2)+B$
1060 N=N1
1070 NEXT D
1080 RETURN
2000 REM****BINARY CONVERSION S/R***
2010 IF NM<256 THEN N=NM:GOSUB 1000:N$=S
$+B$:RETURN
2020 HI=INT(NM/256):LO=NM-256*HI
2030 N=HI:GOSUB 1000:N$="= "+B$
2040 N=LO:GOSUB 1000:N$=N$+" "+B$
2050 RETURN
3000 REM****HEX BYTE S/R**********
3010 HB=INT(N/16):LB=N-HB*16
3020 B$=MID$(X$,HB+1,1)+MID$(X$,LB+1,1)
3030 RETURN
4000 REM****HEX CONVERSION S/R******
4010 IF NM<256 THEN N=NM:GOSUB 3000:H$="
=   "+B$:RETURN
4020 HI=INT(NM/256):LO=NM-256*HI
4030 N=HI:GOSUB 3000:H$="= "+B$
4040 N=LO:GOSUB 3000:H$=H$+" "+B$
4050 RETURN
```

Acorn B

Folgende Zahlen müssen für den Acorn B geändert werden:

```
60 CLS:@%=5
210 PRINT TAB(0);NU;TAB(5);N$;
1050 B$=STR$(R)+B$
```

Dieses Programm berücksichtigt die Besonderheiten, die der Acorn für die Handhabung von numerischen Werten bietet, nicht. Man könnte das Listing deshalb auch in einer Kurzform schreiben.

Spectrum

```
10 REM*******SPECTRUM*********
40 LET S$="=          ":LET X$="0
123456789ABCDEF"
50 REM   S$ CONTAINS 9 SPACES
60 CLS
70 PRINT "  TO DISPLAY DECIMAL N
UMBERS"
80 PRINT " AND THEIR BINARY EQUI
VALENTS"
90 PRINT:PRINT " *******ENTER 0
TO QUIT********":PRINT
100 FOR K=1 TO 1
110 FOR L=1 TO 1
120 INPUT"TYPE ANY POSITIVE WHOL
E NUMBER ";A$
130 LET NU=VAL(A$)
140 IF NU=0 THEN PRINT "PROGRAM
EXIT":STOP
150 IF INT(NU)<>ABS(NU) THEN LET
 L=0
160 IF NU>65535 THEN PRINT NU;"
IS TOO BIG":LET L=0
170 NEXT L
200 LET NM=NU:LET H$="":GOSUB 20
00
210 PRINT NU;TAB(5);N$;
220 IF A$(LEN A$)="+" THEN GOSUB
 4000
230 PRINT H$:PRINT:PRINT
240 LET K=0:NEXT K
300 END
1000 REM**BINARY BYTE S/R**
1010 LET B$=""
1020 FOR D=8 TO 1 STEP-1
1030 LET N1=INT(N/2)
1040 LET R=N-2*N1
1050 LET B$=STR$(R)+B$
1060 LET N=N1
1070 NEXT D
1080 RETURN
2000 REM**BINARY CONVERS S/R**
2010 IF NM<256 THEN LET N=NM:GOS
UB 1000:LET N$=S$+B$:RETURN
2020 LET HI=INT(NM/256):LET LO=N
M-256*HI
2030 LET N=HI:GOSUB 1000:LET N$=
"= "+B$
2040 LET N=LO:GOSUB 1000:LET N$=
N$+" "+B$
2050 RETURN
3000 REM****HEX BYTE S/R******
3010 LET HB=INT(N/16):LET LB=N-H
B*16
3020 LET B$=X$(HB+1)+X$(LB+1)
3030 RETURN
4000 REM***HEX CONVERS S/R****
4010 IF NM<256 THEN LET N=NM:GOS
UB 3000:LET H$="=   "+B$:RETURN
4020 LET HI=INT(NM/256):LET LO=N
M-256*HI
4030 LET N=HI:GOSUB 3000:LET H$=
"= "+B$
4040 LET N=LO:GOSUB 3000:LET H$=
H$+" "+B$
4050 RETURN
```

Wenn Sie eine Zahl mit einem „+" am Ende eingeben (beispielsweise 6435+), wird außer dem dezimalen und dem binären Wert auch die entsprechende Hexzahl angezeigt.

Speicheradressen

Im Vergleich zu unserem vertrauten Dezimalsystem sieht das Hexadezimalsystem umständlich und kompliziert aus. In Verbindung mit dem Acht-Bit-System des Rechners zeigt es sich jedoch als äußerst brauchbar und leicht verständlich.

An diesem Punkt unseres Kurses über Maschinensprache lohnt es sich, auf die Darstellung von Zahlen genauer einzugehen. Es wurde bereits festgestellt, daß die gleichen Zahlen in den verschiedenen Systemen mit unterschiedlichen Methoden dargestellt werden. Mathematische Vorgänge haben jedoch immer die gleichen Ergebnisse, unabhängig von Sprache oder System, in dem die einzelnen Bestandteile eines mathematischen Ausdrucks beschrieben werden.

Der Wert der Zahl 5 bleibt gleich, egal ob sie als Fünf oder Five bezeichnet wird. Genauso verändert sie sich auch nicht, wenn sie 5 oder 101b genannt wird. (Das b hinter 101 bedeutet, daß die Zahl als Binärzahl verstanden werden muß.) Der Grund für die Verwendung eines Zahlensystems ergibt sich aus der Rechengewohnheit oder der Brauchbarkeit des Systems für einen definierten Anwendungsbereich.

Wir empfinden das Dezimalsystem als praktisch, weil wir in unserem täglichen Leben ständig damit umgehen. Das Dezimalsystem ist aber längst nicht das einzige Zahlensystem, das wir wie selbstverständlich einsetzen. So haben zum Beispiel Digitaluhren ein kombiniertes mathematisches System: teilweise dezimal, teilweise Modulo 60 (resultierend auf der Basis 60 Minuten bzw. Sekunden) und teilweise Modulo 24 (die 24 Tagesstunden). Ein weiteres Beispiel: Vor 1971 hatte englisches Geld folgende Einheiten: Ein Shilling enthielt zwölf Pence und ein Pfund 20 Shilling. Obwohl die Beherrschung dieses Systems Schwierigkeiten mit sich brachte, versuchte niemand, mit Pence und Shilling in Form von Dezimalbruchteilen des Pfundes zu rechnen.

Die Zahlensysteme

Für Computer ist das Binärsystem von Vorteil, da sich die internen elektrischen Vorgänge (Folgen der Schaltstellungen EIN und AUS = 1 und 0) damit am besten darstellen lassen. Wenn wir es dabei immer nur mit einzelnen Bytes zu tun hätten, wäre das Binärsystem die perfekte Darstellungsmethode. Im Normalfall sind jedoch besonders Speicheradressen und auch andere oft verwandte Zahlen zu groß, um noch in ein Byte zu passen. Nach jahrelangem Arbeiten mit dem Binärsystem entwickelten Programmierer und Systemingenieure daher ein Zahlensystem, das die logischen Vorteile des Binärcodes hatte, aber wie das Dezimalsystem leicht zu überblicken war. Es entstanden zwei Zahlensysteme: das Hexadezimal- und das Oktalsystem. Das Hexadezimalsystem – auch „Hex" genannt – entwickelte sich zum Standard der Microcomputer. Das auf acht aufbauende Oktalsystem wurde meistens auf Großrechnern eingesetzt, wird jetzt aber mehr und mehr von dem Hexadezimalsystem verdrängt.

Die Basiszahl legt die Anzahl der einstelligen Zahlensymbole des Systems fest und ist außerdem der Multiplikationsfaktor für die einzelnen Stellenwerte. So gibt es z. B. im Dezimalsystem zehn einstellige Zahlensymbole (0–9), und der Wert der einzelnen Stellen erhöht sich von rechts nach links jeweils um den Faktor Zehn.

Das Hexadezimalsystem verfügt über 16 einstellige Zahlensymbole: die Zahlen 0 bis 9 und die Buchstaben A bis F. Beim Aufwärtszählen in Hex werden alle einstelligen Zahlensym-

Dezimal	Binär	Hex
0	00000000	0
1	00000001	1
2	00000010	2
3	00000011	3
7	00000111	7
8	00001000	8
9	00001001	9
10	00001010	A
11	00001011	B
12	00001100	C
13	00001101	D
14	00001110	E
15	00001111	F
16	00010000	10
17	00010001	11
24	00011000	18
25	00011001	19
26	00011010	1A
27	00011011	1B
31	00011111	1F
32	00100000	20
33	00100001	21

Diese Tabelle zeigt die Dezimalzahlen 0 bis 33 mit ihren binären und hexadezimalen Äquivalenten.

bole eingesetzt. Dann geht man auf mehrstellige Darstellungen über, bei denen wie im Dezimalsystem der Wert einer Zahl von seiner Position bestimmt wird. Die Hexzahlen nach 9 sind daher A (dezimal 10), B (dezimal 11), C (dezimal 12), D (dezimal 13), E (dezimal 14) und F (dezimal 15). Da jetzt alle einstelligen Zahlensymbole einmal eingesetzt wurden, ist die auf F folgende Hexzahl 10 (gesprochen: „Eins-Null Hex" = dezimal 16). Der Wert einer Stelle im Hexadezimalsystem erhöht sich bei jeder Bewegung nach links jeweils um den Faktor 16. Im Dezimalsystem bezeichnen wir die einzelnen Stellen als Einer, Zehner, Hunderter und Tausender. Im Hexadezimalsystem repräsentieren die entsprechenden Stellen Einer, Sechzehner, Zweihundertsechsundfünfziger und Viertausendsechsundneunziger.

Zahlenumwandlungen

Wenn Sie jetzt die Binärstellen mit den Hexstellen vergleichen, können Sie leicht den Hauptvorteil der Hexzahlen feststellen: Eine vierstellige Binärzahl läßt sich durch eine einstellige Hexzahl darstellen.
Der Bereich einer einfachen Acht-Bit-Zahl (ein Byte), die im Binärsystem aus acht Stellen und im Hexsystem aus zwei Stellen besteht, ist daher:

```
      0 bis 255        dezimal
00000000 bis 11111111 binär
     00 bis FF        hex
```

Für die Umwandlung einer Hexzahl in eine Binärzahl muß jede Stelle der Hexzahl als vierstellige Binärzahl dargestellt werden. Dabei entsprechen die linken vier Stellen der Binärzahl der äußersten linken Ziffernstelle der Hexzahl und die vier rechten binären Bits der rechten Stelle der Hexzahl. Eine derartige Aufteilung eines Bytes ergibt zwei „Nybbles" (ein Nybble ist die Hälfte eines Bytes). Das Nybble auf der linken Seite des Bytes wird „höherwertiges Nybble" genannt, während man das rechte als das niederwertige Nybble bezeichnet. Hier ein Beispiel:

```
höherwertiges          niederwertiges
    Nybble                 Nybble
       ↓                      ↓
206  =  1100              1110  =  C E
 ↑                        ↑        ↑
dezimal                  binär    hexadezimal
```

Im Hexadezimalsystem lassen sich die aus acht Bits bestehenden Bytes weitaus einfacher berechnen als im Binärsystem. Eine Vertrautheit mit dem System wird sich schnell einstellen, wenn Sie Ihre ersten Erfahrungen mit Hexzahlen, Speicheradressen und Speicherinhalten machen.

In dieser Folge haben wir Programme für den Acorn B, den Commodore 64 und den Spectrum abgedruckt, mit denen sich Byteinhalte des Speichers darstellen lassen. Diese Programme fragen zuerst nach der Anfangsadresse (d. h. der Nummer des ersten Bytes) und dann nach der Anzahl der Bytes, die Sie sich ansehen wollen. Wenn Sie z. B. Byte 1953 als Anfangspunkt angegeben haben und die Inhalte der nächsten vier Bytes abfragen wollen, zeigt die linke Spalte die Dezimalzahl 1953, während der Inhalt von Byte 1953, Byte 1954, Byte 1955 und Byte 1956 in den vier rechten Spalten erscheint.

Wenn die Maschine anzeigt, daß das Byte 1956 die Dezimalzahl 175 enthält, sollten wir uns vor Augen führen, was das eigentlich bedeutet: In einem der Speicherchips existiert ein Bereich, den die Maschine als Byte 1956 bezeichnet und der acht elektrische Zustände enthält. Wenn wir 0 Volt als 0 darstellen und 5 Volt als 1, dann enthält Byte 1956 das Voltmuster 1010111. Wir entscheiden uns dafür, dieses Muster als Binärzahl zu interpretieren, deren dezimales Äquivalent 175 beträgt.

Der Bildschirm kann den „echten" Inhalt eines Bytes nicht anzeigen. Wir sehen nur Zeichen, die wir den Voltmustern der einzelnen Bytes zugeordnet haben: Wir interpretieren die Voltmuster als Binärzahlen, die der Rechner wiederum in Dezimalzahlen darstellt. Nun gehen wir einen Schritt weiter und wandeln die Dezimalzahlen in ASCII-Zeichen (American Standard Code for Information Interchange) um. Diese Zeichen werden in der Spalte am rechten Bildschirmrand dargestellt.

ASCII-Codierung

ASCII ist ein international anerkannter Code, mit dem die meisten Computer arbeiten und der allen Zeichen der Tastatur (ursprünglich der des Teleprinters) Dezimalzahlen zwischen 0 und 127 zuordnet. Dabei entspricht die Zahl 65 dem großen „A", 66 bedeutet „B", 67 „C" etc. Bei den nicht im Alphabet enthaltenen Zeichen entspricht 32 dem Leerzeichen, 42 dem Stern, 13 der Return-Taste usw.

Drucken lassen sich die ASCII-Zeichen zwischen 32 und 127, alle anderen Codes sind entweder nicht fest definiert, Funktionszeichen oder nur auf bestimmten Maschinen vorgesehen. In den Demonstrations-Programmen erscheint auf dem Bildschirm daher für jedes Byte, dessen Codezahl außerhalb des druckbaren Bereiches liegt, ein Stern. In der nächsten Folge werden wir den vollständigen ASCII-Zeichensatz (0 bis 127) abdrucken.

Für das Verständnis des Maschinencodes stellt der ASCII-Zeichensatz eine wichtige Hintergrundinformation dar: Zunächst wird damit

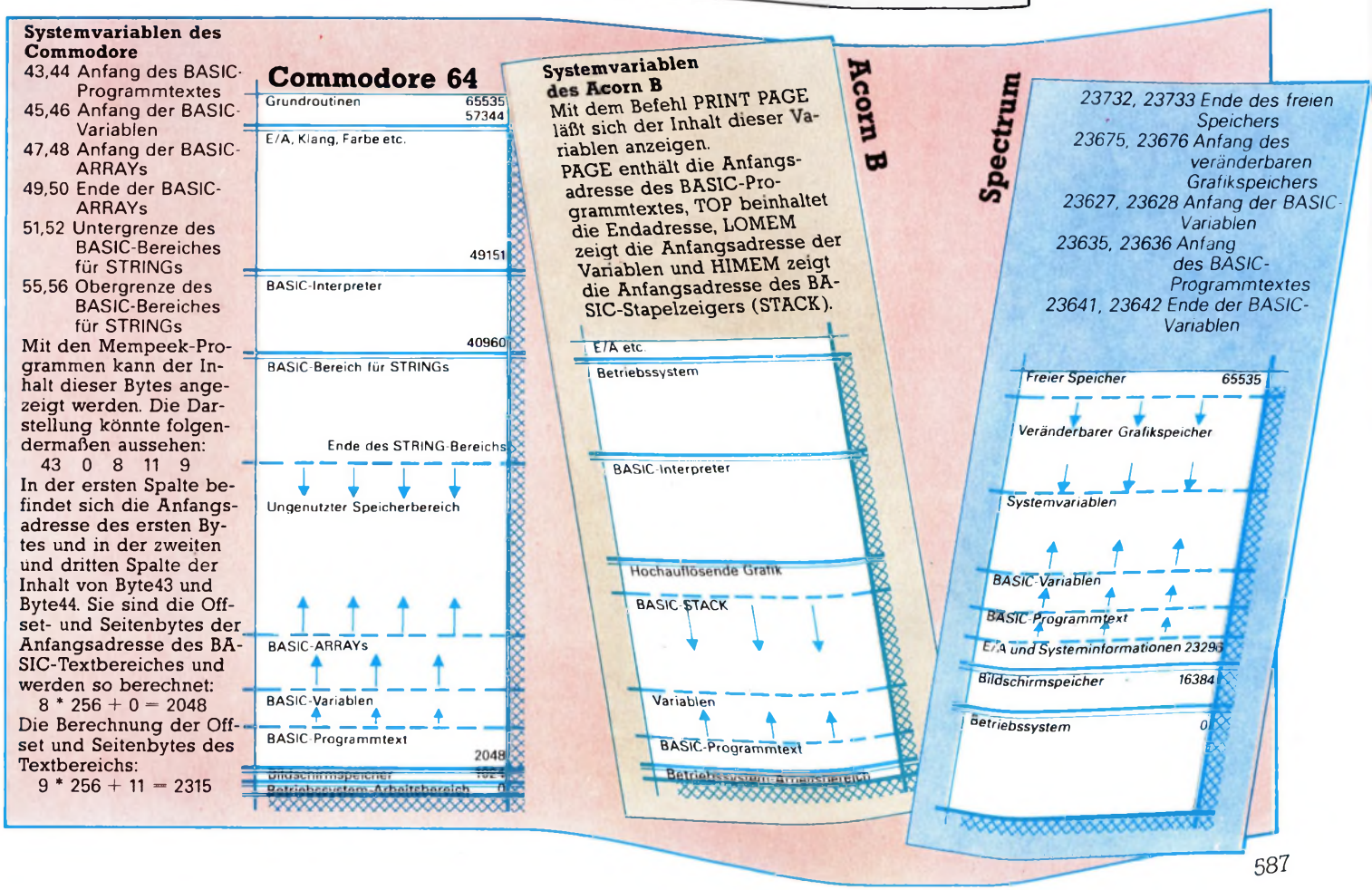
klar, daß der Inhalt eines Speichers unterschiedlich interpretiert werden kann. Sie können den Inhalt eines Bytes als Zahl behandeln, als Zeichen eines Codes oder als Befehl. Weiterhin gewinnen Sie dadurch eine klarere Vorstellung von den Speicherbereichen, die Zeichen enthalten können und vom Betriebssystem der Maschine oder von Ihnen genutzt werden.

Alle Meldungen des Betriebssystems – z. B. Fehlermeldungen, READY, oder START TAPE THEN PRESS RETURN, kurz alles was auf dem Bildschirm erscheint – müssen in ASCII codiert und im Speicher untergebracht sein. Wir erhalten damit einen Eindruck von den Grenzen einer „intelligenten" Maschine. Unsere Intelligenz arbeitet anders: Wir brauchen nicht alle verfügbaren Meldungen zu speichern, wir fassen einen Gedanken und formulieren dann die Worte, die diesen Gedanken ausdrücken.

Memory Maps

Eine Memory Map (Speicheratlas) zeigt in Diagrammform, in welche Bereiche der Speicher aufgeteilt ist und welche Adressen die einzelnen Bereichsgrenzen haben. Einige Speicherbereiche führen immer die gleichen Aufgaben aus. Zum Beispiel werden auf dem Commodore 64 die Bytes 0 bis 1024 von den Variablen des Betriebssystems belegt. Andere Bereiche übernehmen unterschiedliche Aufgaben – abhängig von der Größe und Art des Programms. Die Grenzen sind entweder festgelegt (in unseren Diagrammen als durchgehende Linien angezeigt) oder fließend (gebrochene Linien). Feste Grenzen verändern sich nicht, während die Größe der Bereiche mit fließenden Grenzen je nach Bedarf variiert. In der Memory Map des Commodore 64 sind die Grenzen des Bildschirmspeichers auf Byte 1024 bis 2048 festgelegt, während der Bereich für BASIC-Variablen von der Anzahl der eingesetzten Variablen bestimmt wird.

Mit den Mempeek-Programmen lassen sich auch die aktuellen Grenzen der variablen Speicherbereiche anzeigen. Der Commodore besitzt sechs Zeiger (Pointer) für die Systemvariablen. Wir haben diese Variablen in unseren Beispielen unten aufgeführt und können aus den Bytepaaren die einzelnen Speicheradressen erfahren. Das BBC-BASIC hat übrigens vier Systemvariablen und der Spectrum fünf.



Acorn B

```
7 REM*****************************
8 REM*     BBC        MEMPEEK 1    *
9 REM*****************************
20 MODE 7
30 *TV 255
40 CLS
50 REPEAT
100 INPUT"START ADDRESS ",SA
200 INPUT"NUMBER OF BYTES (0 TO QUIT)",B
N
250 PRINT "*****************************
**********"
300 FOR B%=SA TO (SA+BN-1) STEP 4
350 H$="":@%=6
400 PRINT TAB(0);B%;TAB(8);
450 @%=4
500 FOR C=0 TO 3
550 PK%=?(B%+C):PK$="."
600 PRINT PK%;
650 IF PK%=13 THEN PK$=CHR$(124)
700 IF (PK%>31) AND (PK%<128) THEN PK$=C
HR$(PK%)
750 H$=H$+PK$
800 NEXT C
850 PRINT TAB(32);H$
900 NEXT B%
950 UNTIL BN=0
1000 REM*************************
```

Spectrum

```
7 REM*************************
8 REM*  SPECTRUM  MEMPEEK 1   *
9 REM*************************
30 DIM H$(4)
50 FOR L=0 TO 1 STEP 0
100 INPUT"START ADDRESS ";SA
200 INPUT"NUMBER OF BYTES (0 TO
QUIT)";BN
250 PRINT "*********************
*********"
300 FOR B=SA TO (SA+BN-1) STEP 4
350 LET H$="...."
400 PRINT B;TAB 7;
500 FOR C=0 TO 3
550 LET PK=PEEK(B+C)
600 PRINT PK;" ";
650 IF (PK>31) AND (PK<128) THEN
  LET H$(C+1)=CHR$ PK
700 IF PK=13 THEN LET H$(C+1)="■
"
800 NEXT C
850 PRINT TAB 26;H$
900 NEXT B
950 IF BN=0 THEN LET L=2
1000 NEXT L
1050 REM*************************
```

Commodore 64

```
7 REM*************************
8 REM*    COMMODORE MEMPEEK 1   *
9 REM*************************
30 PRINT CHR$(147)     :REM CLEAR SCREEN
40 PRINT CHR$(142)     :REM UPPER CASE
50 FOR LP=0 TO 1 STEP 0
100 INPUT"START ADDRESS ";SA
200 INPUT"NUMBER OF BYTES (0 TO QUIT)";B
N
250 PRINT "*****************************
**********"
300 FOR B=SA TO (SA+BN-1) STEP 4
350 H$=""
400 PRINT B;TAB(8);
500 FOR C=0 TO 3
550 PK=PEEK(B+C):PK$="."
600 PRINT TAB(8+5*C);PK;
650 IF PK=0 THEN PK$=CHR$(122)
700 IF (PK>31) AND (PK<128) THEN PK$=CHR
$(PK)
750 H$=H$+PK$
800 NEXT C
850 PRINT TAB(32);H$
900 NEXT B
950 IF BN=0 THEN LP=1
1000 NEXT LP
1050 REM*************************
```

Wie funktionieren die „Mempeek"-Programme?

Bevor Sie die Mempeek-Programme ausprobieren, sollten Sie diese mit SAVE speichern, da Schreibfehler zu einem Abbruch des gesamten Systems führen können.

Das Programm fragt zunächst nach der Anfangsadresse und dann nach der Anzahl der Bytes, die Sie sich ansehen wollen. Beide Eingaben sollten ganze Zahlen zwischen 0 und 65535 sein. Die Eingabe von 0 bei der Byteanzahl beendet das Programm. Wenn Sie z. B. die Anfangsadresse 230 eingeben, kann der Bildschirm folgendes anzeigen:

```
ANFANGSADRESSE? 230
ANZAHL BYTES (0 = ENDE) ? 8

230 193 32 65 49 . A1
234 129 64 93 98 . @]b
```

Die linke Spalte zeigt die Dezimaladresse des ersten Bytes an, die nächsten vier Spalten bringen den Inhalt der vier Bytes von dieser Adresse an als Dezimalzahlen auf den Schirm, während die rechte Spalte das dem Byteinhalt entsprechende Zeichen (wenn möglich) darstellt. Läßt sich das Zeichen nicht darstellen, wird ein „." angezeigt.

Beginnen Sie am besten damit, im Speicher ein wenig „herumzuwandern", sehen Sie sich die interessanten Adressen an und versuchen Sie, die Meldungen des Betriebssystems und die Schlüsselworte des BASIC zu finden. Das Benutzerhandbuch kann Ihnen dabei helfen.

Wenn Sie die Zeiger gefunden haben, die die Grenzen der Speicherbereiche angeben, dann können Sie in das Programm ein paar REM-Zeilen einfügen und die Auswirkungen auf die Zeigerwerte beobachten. Fügen Sie auch am Anfang des Programms einige Zeilen für die STRING-Bearbeitung ein und untersuchen Sie, wie sich dies auf die Zeiger und den Inhalt des Speicherbereichs für Variablen auswirkt.

Texte und Zeichen

In dieser Folge wird untersucht, wie BASIC-Programme im Arbeitsspeicher untergebracht und vom BASIC-Interpreter ausgeführt werden. Diese Informationen geben wesentlichen Aufschluß darüber, wie der Maschinencode den Speicher handhabt.

Wie Sie bereits wissen, ist von dem Augenblick an, in dem Sie Ihr Gerät einschalten, ein hochentwickeltes Programm aktiv: das Betriebssystem. Es ist in einigen ROM-Chips im Inneren des Computers eingebrannt und steuert die Grundfunktionen der Maschine. Beispielsweise bringt das Betriebssystem das Bild auf den Monitor, handhabt Druckerausgabe und Diskettenspeicherung und überprüft die Tastatur auf Eingaben. Das Betriebssystem sieht alle einkommenden Daten als Impulse an, die es verarbeiten muß.

Eines der Betriebssystemmodule ist der BASIC-Interpreter, mit dem die Texte von BASIC-Programmen analysiert und ausgeführt werden. Bei der Eingabe erkennt das Betriebssystem an der Zeilennummer (am Anfang jeder neuen Zeile), daß ein BASIC-Programm vorliegt. Es bringt die Zeichen der Programmzeilen (mit wenigen Ausnahmen) dann als einzelne Bytes in einem speziell für BASIC-Programmtexte reservierten Speicherbereich unter. Der Interpreter wiederum versteht BASIC-Programme nur als Daten, die verarbeitet werden müssen. Bei der Eingabe von RUN wird ihm vom Betriebssystem die Steuerung übergeben, und er bearbeitet nun selbständig die BASIC-Daten.

Der Interpreter verändert das Programm nicht, sondern übersetzt es in die Maschinensprache und läßt es ablaufen. Da er allen Befehlen gehorcht, können Sie auch den Speicherbereich abfragen, der das Programm enthält. Wenn möglich führt der Interpreter die Programmbefehle aus. Ist dies nicht machbar, meldet er SYNTAX ERROR, OVERFLOW ERROR oder ähnliches.

Mit Ausnahme der Schlüsselworte werden BASIC-Programme vom Betriebssystem zeichenweise gespeichert. Wird die Buchstabenkombination (oder auch die Zeichen, Zahlen oder elektrischen Impulsfolgen) eines Schlüsselwortes erkannt, dann tauscht das Betriebssystem dieses Wort gegen eine Codezahl mit der Länge eines Bytes aus, die „Token" (Ersatzzeichen) genannt wird. Dieser Vorgang spart Speicherplatz (der Befehl RESTORE beispielsweise würde sonst sieben Zeichen belegen) und vereinfacht somit die Arbeit des Interpreters.

Obwohl alle Maschinentypen unterschiedliche Tokensysteme verwenden, sind den Token jedoch fast immer Zahlenwerte über 127 zugeordnet. Die ASCII-Codezahlen für druckbare Zeichen liegen zwischen 32 und 127. Trifft der Interpreter auf ein Token-Byte, ruft er einfach die entsprechende Unterroutine auf.

Wenn Sie LIST aufrufen, um Ihr Programm anzusehen, untersucht das Betriebssystem die einzelnen Bytes des Textbereiches. Trifft es auf einen Wert über 127, dann interpretiert es diesen Wert als Token. Im Speicher befindet sich ebenfalls eine Liste aller BASIC-Schlüsselworte in ASCII-Darstellung. Der Tokenwert zeigt auf eines dieser Worte, und das Betriebssystem setzt es anstelle des Tokens für die Bildschirmdarstellung des Programms ein.

ASCII-Codes, größer als 127, können jedoch auf einigen Computern im Original dargestellt werden, wenn sie in Anführungszeichen eingeschlossen sind. Ohne Anführungszeichen werden sie als Token interpretiert.

Die Zeilenlänge

Generell enthalten die ersten drei oder vier Bytes einer BASIC-Programmzeile die Zeilennummer und Informationen über die Zeilenlänge (siehe Bild auf der nächsten Seite). Das Format kann sich allerdings von Computer zu Computer unterscheiden. Die Zeilennummer wird dabei nicht als ASCII-Code gespeichert (beispielsweise würde die Zeile 61030 allein für ihre Nummer schon fünf Bytes benötigen), sondern als eine aus zwei Bytes bestehende Ganzzahl (Integer). Den Zahlen von 0 bis 255 (die sich in einem einzigen Byte unterbringen lassen) wird dabei ein Byte mit dem Wert Null vorangestellt, um das Zwei-Byte-Format zu erhalten. Größere Zahlen werden ähnlich wie bei der Seitenadressierung gespeichert: Das erste Byte wird mit 256 multipliziert und mit dem zweiten Byte addiert. Die Zahl 1000 wird daher als 3232 (3 * 256 + 232 = 1000) gespeichert. Diese beiden Bytes befinden sich in Programmzeilen auf immer der gleichen Stelle, die jedoch von System zu System variieren kann.

Die Information über die Zeilenlänge ist auf dem Acorn B in einem einzigen Byte und auf dem Spectrum in zwei Bytes gespeichert. Der Inhalt dieser Bytes gibt die Zahl der in der Zeile enthaltenen Bytes einschließlich der Bytes für Zeilennummer und Länge an. Ist die

Die Codes 32 bis 127 enthalten die druckbaren ASCII-Zeichen, die (mit wenigen Ausnahmen) auf allen Computern identisch sind. In den Handbüchern der einzelnen Maschinen sind die zugehörigen ASCII-Codes genau beschrieben.

ASCII CODE	ASCII	ASCII CODE	ASCII
32	Space	80	P
33	!	81	Q
34	"	82	R
35	#	83	S
36	S	84	T
37	%	85	U
38	&	86	V
39	'	87	W
40	(	88	X
41	)	89	Y
42	*	90	Z
43	+	91	[
44	,	92	\
45	-	93	]
46	.	94	^
47	/	95	_
48	0	96	`
49	1	97	a
50	2	98	b
51	3	99	c
52	4	100	d
53	5	101	e
54	6	102	f
55	7	103	g
56	8	104	h
57	9	105	i
58	:	106	j
59	;	107	k
60	<	108	l
61	=	109	m
62	>	110	n
63	?	111	o
64	@	112	p
65	A	113	q
66	B	114	r
67	C	115	s
68	D	116	t
69	E	117	u
70	F	118	v
71	G	119	w
72	H	120	x
73	I	121	y
74	J	122	z
75	K	123	{
76	L	124	!
77	M	125	}
78	N	126	~
79	O	127	Delete

Adresse des ersten Bytes einer BASIC-Programmzeile bekannt, brauchen Sie zu diesem Wert nur den Inhalt des Bytes der Zeilenlänge zu addieren, um die Anfangsadresse der nächsten Programmzeile zu erhalten. Da ein Byte nur Zahlen bis 255 speichern kann, lassen sich in einer Programmzeile des Acorn B auch nur 255 Zeichen unterbringen. Versuchen Sie einmal, mit dem Programm Mempeek festzustellen, ob diese Grenze sich auf die Anzahl der Zeichen bezieht, die Sie in einer Programmzeile eingeben können, oder auf die Länge der Zeile, die gespeichert wird.

Auf dem Commodore wurde das Byte für die Zeilenlänge durch eine aus zwei Bytes bestehende „Verbindungsadresse" (Link Address) ersetzt. Diese Verbindungsadresse enthält die Anfangsadresse der nächsten Programmzeile.

Dabei ist interessant, daß der Acorn B und der Spectrum sich die Anfangsadresse der nächsten Zeile aus der aktuellen Adresse und der Zeilenlänge errechnen (ein Verfahren, das zwar langsam ist, aber ein Byte einspart). Der Commodore dagegen speichert diese Adresse und benötigt dafür ein zusätzliches Byte, ist aber schneller. Dieses Beispiel zeigt, daß es keine absolute Formel für die Konstruktion eines Computers gibt, sondern nur die individuellen Methoden der einzelnen Designer. Das spiegelt auch die Überlegungen der Computer-Konstrukteure wider, die bei der Planung eines Gerätes zwischen zwei grundlegenden Spezifikationen wählen müssen: entweder eine schnelle, jedoch teure Maschine

ASCII-CODE	ASCII	Commodore	Spectrum	Acorn B
0	NUL – keine Auswirkung	–	–	Null
1	SOH – Startsignal einer Zeichenfolge	–	–	Nächstes Zeichen zum Drucker senden
2	STX – Start einer Zeichenfolge	–	–	Drucker anschalten
3	ETX – Ende einer Zeichenfolge	–	–	Drucker ausschalten
4	EOT – Ende der Übertragung	–	–	Text/Grafikcursor trennen
5	ENQ – Signalanfrage	"Weiß"-Taste	–	Text/Grafikcursor verbinden
6	ACK – Zeichen empfangen	–	PRINT	VDU-Treiber einschalten
7	BEL – Klingelzeichen	–	EDIT	kurzes Klingelzeichen
8	BS – Backspace	CBM-Taste abschalten	Cursor links	Backspace Cursor
9	HT – Horizontaler Tabulator	CBM-Taste anschalten	Cursor rechts	Vorwärtsschritt des Cursors
10	LF – Zeilenvorschub	–	Cursor eine Zeile nach unten	Cursor eine Zeile nach unten
11	VT – Vertikaler Tabulator	–	Cursor eine Zeile nach oben	Cursor eine Zeile nach oben
12	FF – Formularvorschub	–	Löschtaste	Textbereich löschen
13	CR – Wagenrücklauf	RETURN	ENTER	Return
14	SO – Dauerumschaltung	Kleinbuchstaben einschalten	Zahl	Seitenschaltung ein
15	SI – Rückschaltung	–	–	Seitenschaltung aus
16	DLE – Datenübertragungsumschaltung	–	INK	Grafikbereich löschen
17	DC1 – Gerätesteuerung 1	Cursor eine Zeile nach unten	PAPER	Textfarbe definieren
18	DC2 – Gerätesteuerung 2	Umgekehrte Zeichendarstellung an	FLASH	Grafikfarbe definieren
19	DC3 – Gerätesteuerung 3	Cursor auf HOME-Position	BRIGHT	logische Farbe definieren
20	DC4 – Gerätesteuerung 4	Löschtaste	INVERSE	auf Systemeinstellung der logischen Farbe zurückstellen
21	NAK – Zeichen nicht empfangen	–	OVER	VDU-Treiber abschalten
22	SYN – Synchronisierung	–	AT	Darstellungsart auswählen
23	ETB – Ende eines Datenblocks	–	TAB	dargestelltes Zeichen umprogrammieren
24	CAN – ungültig	–	–	Grafikfenster definieren
25	EM – Ende der Aufzeichnung	–	–	Plot m,x,y
26	SUB – Zeichen ersetzen	–	–	auf Systemeinstellung der Fenster zurückstellen
27	ESC – Codeumschaltung	–	–	Null
28	FS – Filetrennung	"Rot"-Taste	–	Textfenster definieren
29	GS – Gruppentrennung	Cursor ein Zeichen nach rechts	–	Grafikmode definieren
30	RS – Untergruppentrennung	"Grün"-Taste	–	bewege Textcursor
31	US – Teilgruppentrennung	"Blau"-Taste	–	Textcursor auf Koordinaten x,y stellen

Der „American Standard Code for Information Interchange" (Amerikanische Standardcodierung für den Informationsaustausch) ordnet jedem Standardzeichen einen Zahlencode zwischen 0 und 127 zu. Die Zeichen der Codes von 0 bis 31 lassen sich nicht drucken, werden aber als Steuersignale für den Betrieb von Peripheriegeräten wie Bildschirm und Drucker eingesetzt. Die Bedeutung dieser Codes ist daher von Maschine zu Maschine verschieden – wie aus der Tabelle ersichtlich. Viele Geräte – in unserer Tabelle der Commodore und der Spectrum – nutzen einen Teil der Codes nicht (mit -- gekennzeichnet).

oder ein langsames, aber preiswertes Gerät. Das Optimum ergibt sich natürlich aus der Verbindung beider positiver Eigenschaften. Vor eine ähnliche Entscheidung sind auch Sie gestellt, wenn Sie BASIC-Programme auf Maschinen mit begrenztem Speicher (zum Beispiel Grundversionen des VC 20 und ZX81) schreiben und das Verhältnis zwischen Ausführungsgeschwindigkeit und Speicherausnutzung gründlich abwägen müssen.

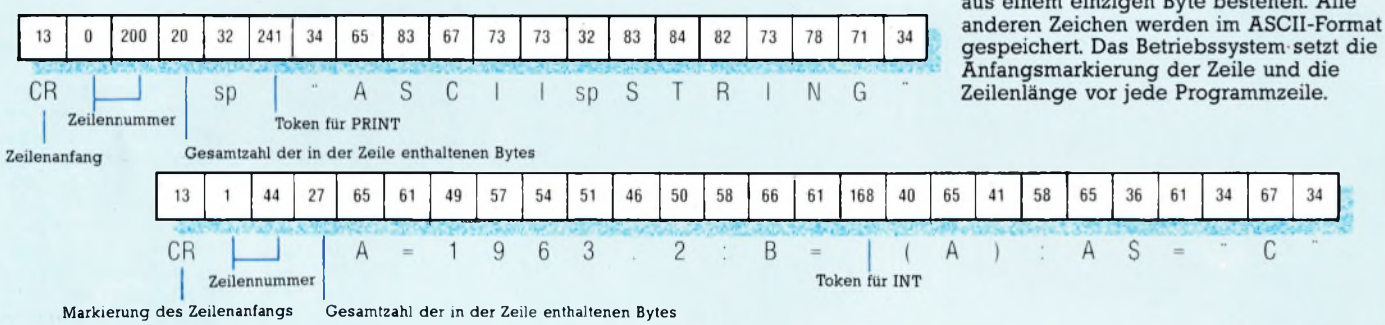
Schließlich hat jede Programmzeile entweder am Anfang oder am Ende eine Markierung. Auf dem Acorn B beginnt jede Zeile mit einem Byte, das die Zahl 13 enthält (ASCII-Bode für „Return" – Wagenrücklauf). Der Spectrum markiert damit das Zeilenende. Das Commodore-BASIC schließt seine Zeilen mit dem „Null"-Byte (ASCII für "") ab.

Die Speicherung von BASIC-Programmen

Jede Maschine speichert BASIC-Texte nach einer eigenen Methode. Im folgenden haben wir drei Beispiele aufgeführt.

Acorn B

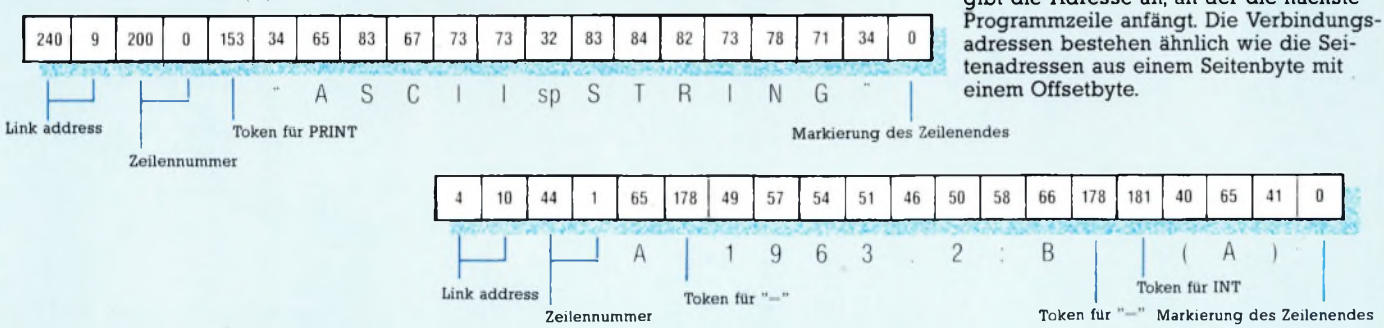
```
200 PRINT"ASCII STRING"
300 A=1963.2:B=INT(A):AS="C"
```



In diesem Beispiel werden die Schlüsselwörter des BASIC von Token ersetzt, die aus einem einzigen Byte bestehen. Alle anderen Zeichen werden im ASCII-Format gespeichert. Das Betriebssystem setzt die Anfangsmarkierung der Zeile und die Zeilenlänge vor jede Programmzeile.

Commodore 64

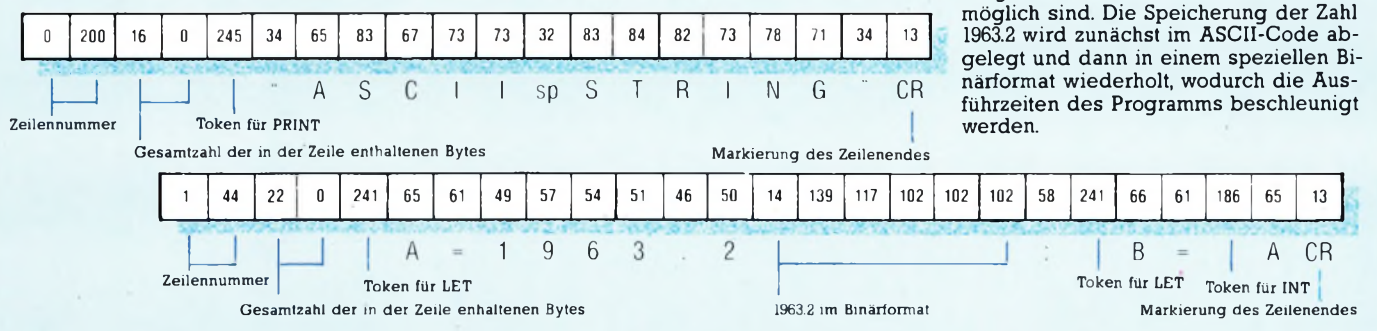
```
200 PRINT "ASCII STRING"
300 A=1963.2:B=INT (A)
```



Die Verbindungsadresse (Link address) gibt die Adresse an, an der die nächste Programmzeile anfängt. Die Verbindungsadressen bestehen ähnlich wie die Seitenadressen aus einem Seitenbyte mit einem Offsetbyte.

Spectrum

```
200 PRINT "ASCII STRING"
300 LET A=1963.2:LET B=INTA
```



Die Zeilenlänge wird hierbei in zwei Bytes statt nur in einem gespeichert, so daß Programmzeilen mit mehr als 255 Zeichen möglich sind. Die Speicherung der Zahl 1963.2 wird zunächst im ASCII-Code abgelegt und dann in einem speziellen Binärformat wiederholt, wodurch die Ausführzeiten des Programms beschleunigt werden.

Befehlsstrukturen

In dieser Folge fassen wir die Standardformate der Maschinencodeprogrammierung zusammen. Darunter fällt unter anderem die Adressierung des niedrigen und hohen Bytes.

Wenn Sie ein Programm mit RUN aufrufen, muß das Betriebssystem erst einmal feststellen, in welchem Bereich des Speichers der Programmtext überhaupt liegt. Es untersucht deshalb zuerst den Zeiger, der die Anfangsadresse des Programmtextes enthält. Dabei stellt sich die Frage, warum das Betriebssystem nicht statt der Pointeradresse die direkte Anfangsadresse speichert.

Der Hauptgrund für diese „Umständlichkeit" ist die Flexibilität. Sie erinnern sich: Das Betriebssystem ist ein im ROM festgelegtes Programm, dessen interne Daten (z. B. Speicheradressen) sich nicht ändern lassen. Nehmen wir an, daß über einen längeren Zeitraum mehrere Versionen eines Computers herausgebracht werden. Bei Version 1 genügte es vielleicht, den Anfang des Speicherbereiches für BASIC-Programmtexte auf die Adresse 2048 zu legen, für Version 2 aber wurde es notwendig, diesen Bereich bei Byte 4096 anfangen zu lassen. Ist eine Adresse jedoch erst einmal fixiert, dann können die neueren Maschinen nicht das Betriebssystem früherer Versionen einsetzen, da der Programmtext nicht an der alten Adresse beginnt. Für jede neue Maschine müßten daher teure, neue ROMs entwickelt werden, und außerdem würde Software, die auf einer Version entwickelt wurde, nicht auf anderen Modellen laufen. Enthält das Betriebssystem jedoch nur eine Zeigeradresse, dann läßt es sich für alle Versionen der Maschine verwenden, wobei nur der Inhalt des Pointers von Version zu Version geändert wird.

Es ist üblich, daß Zeigeradressen das sogenannte „lo-hi"-Format haben. Wenn z. B. Byte 43 und Byte 44 die Adresse 7671 enthalten (Seite 29, Offset 247), dann ist in Byte 43 der Wert 247 (Offset oder niederwertiges Byte der Adresse) gespeichert und in Byte 44 der Wert 29 (Seitenadresse oder höherwertiges Byte der Adresse).

Bei einer Wiederholung des Beispiels mit Hexadezimalzahlen werden die Vorteile dieses Zahlensystems deutlich. (Von nun an werden wir Adressen und andere Zahlen im Hexadezimalformat schreiben. Hexzahlen sind dabei durch ein vorangehendes $ gekennzeichnet.) Der Zeiger besteht aus den Bytes $2B und $2C und zeigt auf die Adresse $1DF7. $2B enthält daher $F7 (das niederwertige Byte der Adresse), während $2C das höherwertige Adressbyte $1D speichert. In einer Hexadresse sind die beiden rechten Hexzahlen das niederwertige Byte, während die linken beiden Zahlen das höherwertige Byte (die Seitenadresse) darstellen.

Weiterhin bezieht man sich bei der Angabe einer Speicheradresse oft nur auf die Adresse, die das niederwertige Byte enthält. Man könnte daher sagen, daß auf dem Commodore 64 Byte 43 den Anfang des BASIC-Programmtextes angibt. Trotzdem ist es selbstverständlich, daß Byte 43 und Byte 44 gemeinsam der Pointer sind.

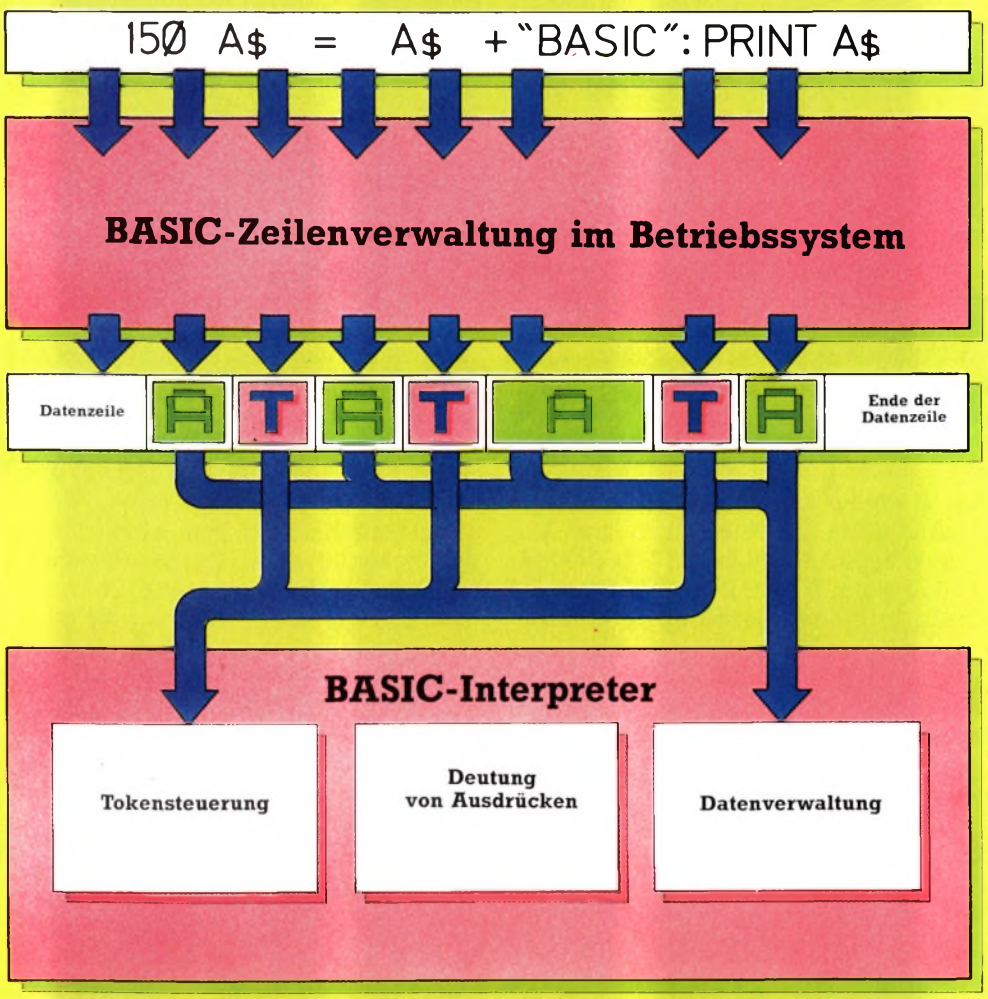
Token werden ebenfalls nach einem bestimmten Standard behandelt. Für die Programmierung im Maschinencode haben sie zwei Bedeutungen: Sie stellen englischsprachige Befehle wie PRINT oder RESTORE im numerischen Code mit einem Byte Länge dar und werden zudem als Offset eingesetzt. Obwohl die meisten BASIC-Befehle aus einem Wort bestehen, sind sie für das Betriebssystem nicht mit einem einzigen Vorgang erledigt. Der Befehl PRINT z. B. erfordert, daß Daten, die gedruckt werden sollen, im Speicher gefunden, bewertet und dann zeichenweise im ASCII-Code zum Bildschirm gesendet werden. Diese unterschiedlichen Aufgaben führt eine Unterroutine des BASIC-Interpreterprogramms aus. Findet der Interpreter in einer Programmzeile den Token für PRINT, dann sucht er mit diesem Wert die entsprechende Unterroutine und führt sie aus.

Einsatz des Tokens

Nehmen Sie an, in einer BASIC-Version gäbe es nur die drei Befehle INPUT, PRINT und STOP, denen jeweils die Token $80, $81 Unterroutinen des Interpreters fangen bei den Bytes $D010, $EA97 und $EC00 an. Diese Adressen sind im lo-hi-Format in sechs Bytes von $FA00 bis $FA05 gespeichert. Die Tabelle enthält also drei Zeiger mit einer Länge von je zwei Bytes. Wenn unser Interpreter jetzt einen Token findet, – z. B. $81 – , dann subtrahiert er davon $80, multipliziert den Rest mit 2 und addiert das Ergebnis zu $FA00. In diesem Fall ergibt das Endresultat $FA02, das das niederwertige Adressbyte des Zeigers der PRINT-Routine enthält. Der BASIC-Befehl PRINT wird also durch den Token $81 ersetzt. Dieser enthält den Offset für eine Zeigertabelle, mit der der Interpreter die eigene Unterroutine findet. Bei den anderen Token liefert dieser Algo-

BASIC

Sie geben dies auf der Tastatur ein:

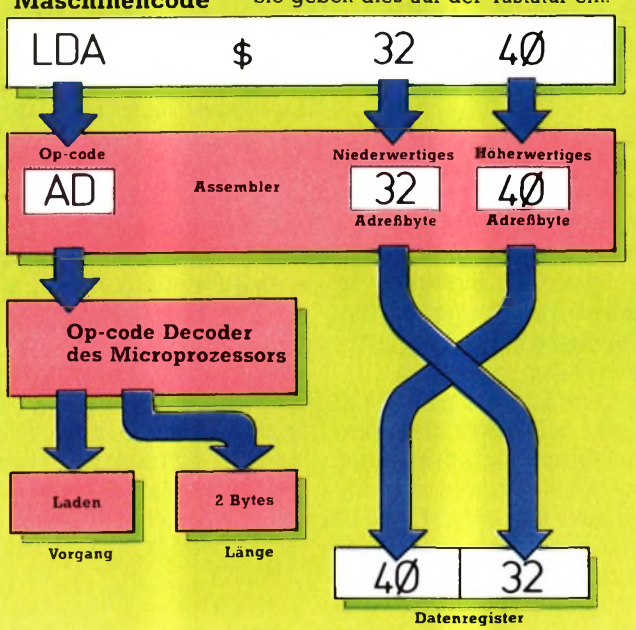


T Token

A In ASCII codierte Daten

Befehle im Maschinencode

Sie geben dies auf der Tastatur ein:



Schritt für Schritt

Diese Darstellung verdeutlicht, wie eine Programmzeile in BASIC und ein Befehl im Maschinencode übersetzt und ausgeführt werden.

Das Betriebssystem gibt das Zeilenformat mit allen Informationen vor und ersetzt die BASIC-Schlüsselworte durch Token.

Sie geben RUN ein.

Der BASIC-Interpreter untersucht die Programmzeile nach Token und den entsprechenden Daten und ruft dann entsprechend dem Wert der Token die jeweiligen Unterroutinen auf.

Der Assembler übersetzt die mnemotischen Codes in den entsprechenden Op-code (1 Byte) und speichert den Operanden (2 Bytes) im lo-hi-Format.

Beim Ausführen des Befehls decodiert der Prozessor den Op-code in einen Operations- und einen Längencode, damit die richtige Anzahl der Bytes als Operand behandelt werden kann.

rithmus ebenfalls die Zeigeradresse der entsprechenden Unterroutine.

Hier wird der Unterschied zwischen BASIC, einer sogenannten „Hochsprache", und dem Maschinencode deutlich. BASIC ist für uns leichter zu verstehen, da es Codeworte enthält, die der englischen Sprache ähnlich sind, und außerdem algebraische Logik, Zahlen und Zeichenketten eingesetzt werden. Wenn wir die Schlüsselworte durch Token ersetzen und den Rest der Zeichen als ASCII-Codes darstellen, dann ähnelt ein Programm schon mehr dem Format, das ein Microprozessor verarbeiten kann.

Ein wichtiger Faktor bei Speicherabläufen ist das Konzept des „Zusammenhangs". Es gibt im Speicherbereich für BASIC-Texte eine ganze Reihe von Codes – ASCII-Codes zur Zeichendarstellung, Token als Kurzform für Befehle und spezielle Binärzahlen für Zahlendarstellung. Alle diese Codes erscheinen im Maschinencode als Binärzahlen von 00000000 bis 11111111 ($00 bis $FF oder 0 bis 255 dezimal). Sie werden in einem Speicherbyte untergebracht und nur aus ihrem Zusamenhang heraus interpretiert. Der Textbereich des Commodore 64 könnte z. B. folgende Programmzeile enthalten:

```
200 rem******left$********
```

Darin ist die Dezimalzahl 200 in drei verschiedenen Bytes enthalten: je einmal als niederwertiges Byte der Verbindungsadresse und der Zeilenadresse und einmal im Token für „left$". Jedes dieser Bytes sieht genauso aus wie die anderen, stellt aber etwas völlig anderes dar. Nur aus dem Zusammenhang wird deutlich, wie die Bytes zu interpretieren sind.

Am Anfang dieser Serie erwähnten wir, daß alle Information als eine Art Maschinencode im Computer untergebracht ist. Die ASCII-Codes und die Token sind uns nun vom Konzept her bekannt. Im weiteren Verlauf werden wir die Programmierung im Maschinencode ausführlich behandeln.

Programme im Maschinencode sind Bytefolgen, die sich an jedem Ort des Speichers befinden können. Sie enthalten eine Mischung aus Prozessorbefehlen und Daten, die der Prozessor verarbeiten soll. Wie bei anderen Speicherbytes lassen sich Daten von Befehlsbytes nur durch den Zusammenhang unterscheiden. Wir werden daher zunächst das Befehlsformat des Maschinencodes erklären.

Ein Befehl im Maschinencode beginnt immer mit einem Code, der den auszuführenden Vorgang genau angibt. Dieser Schlüssel wird „Op-code" oder „opc" genannt und hat die Länge von einem oder zwei Bytes. Ein Opcode kann völlig eigenständig sein. Im Normalfall folgen ihm aber ein oder zwei Datenbytes. Ein einzelnes Byte enthält dabei entweder eine numerische Konstante oder einen ASCII-Code, während zwei Datenbytes immer eine Adresse (Format: niederwertiges Byte/höherwertiges Byte) darstellen. Diese Definition läßt auch sofort die unterschiedliche Arbeitsweise der Microprozessoren deutlich werden: Der Acorn B wird von einem MOS-Tech-6502A gesteuert, der Commodore 64 von einem MOS-Tech-6510 (dem 6502A sehr ähnlich, wir sprechen daher nur von diesem), während in den Spectrum der Z80A von Zilog eingebaut ist. Der 6502 und der Z80 bauen auf dem gleichen Konzept auf, unterscheiden sich aber wesentlich in den Einzelheiten. Ganz besonders die Maschinencodes des Z80 und des 6502 sind völlig verschieden voneinander. So bestehen die Op-codes des 6502 immer aus einem Byte, das von einem, zwei oder keinem Datenbyte gefolgt werden kann, während Op-codes des Z80 zwei Bytes lang sein können, gefolgt von einem, zwei oder keinem Datenbyte.

Das interne Programm eines Microprozessors decodiert einen Op-code in einen Operations- und einen Längencode. Der Längencode gibt ihm die Information, wie er die auf den Op-code folgenden Bytes interpretieren soll. Nehmen wir die folgenden Hexadezimalbytes des 6502:

```
A9 0E 8D 01 4E 60 44 52 41 54
```

Darin sind drei Befehle enthalten, denen vier Bytes im ASCII-Code folgen. Die Bytes lassen sich auch so schreiben:

```
A9 0E
8D 01 4E
60
44
52
41
54
```

Der erste Befehl ist A9, auf den immer ein Datenbyte folgt. Der nächste Befehl ist 8D – immer gefolgt von zwei Datenbytes, während der Op-code 60 keine Daten benötigt, sondern eine Programmverzweigung auslöst, durch die die folgenden Datenbytes von dem Prozessor nicht mehr gelesen werden. Erhält der Microprozessor zuerst das Byte A9, laufen danach alle Befehle problemlos ab. Die in jedem Opc enthaltene Information stellt sicher, daß der Prozessor die richtige Anzahl von Datenbytes nach den Op-codes als Datenbytes versteht und daß das darauffolgende Byte wiederum als Op-code interpretiert wird. Wird dem Prozessor jedoch das zweite Byte (0E) übermittelt, wenn er einen Befehl erwartet, dann behandelt er dieses Byte als Op-code und interpretiert die Bytefolge so:

```
0E 8D 01
4E 60 44
52
```

Dem Opc 0E folgen zwei Datenbytes, ebenso dem Opc 4E, während Opc 52 keinen Op-code darstellt und einen Syntaxfehler verursacht. Damit wird deutlich, wie ein Mißverständnis am Programmanfang eine Kette von logischen Fehlern auslösen kann.

Unser Beispiel zeigt aber auch, wie unbequem Maschinencode (zumindest anfangs) zu lesen und zu schreiben ist. Die einzelnen Befehle unterscheiden sich nur durch ihre Stellung innerhalb des Speichers.

Mnemotische Zeichencodes

Einige dieser Unbequemlichkeiten lassen sich vermeiden, wenn für die Programmentwicklung mnemotische Zeichencodes statt numerischer Op-codes eingesetzt werden. Zum Laden der Programme müssen die Zeichen wiederum in Op-codes übersetzt werden. Die Assemblersprache eines Prozessors verwendet diese mnemotischen Codes, deren Übertragung in die Maschinensprache „Assemblierung" genannt wird. Zwischen einer Folge mnemotischer Zeichencodes und einem Set Op-codes besteht eine unmittelbare Übereinstimmung: Obwohl Assembler eine höhere Programmiersprache ist als der Maschinencode, ist der Unterschied zwischen beiden nur minimal.

Wenn wir die obige Maschinencodefolge in der Assemblersprache des 6502 darstellen, sieht das folgendermaßen aus:

```
0000 A9 0E     LDA #$0E
0002 8D 01 4E  STA $4E01
0005 60        RTS
```

Im Assembler des Z80 sieht die gleiche Folge jedoch völlig anders aus:

```
0000 3E 0E     LD A, $06
0002 32 01 4E  LD($4E01),A
0005 C9        RET
```

Die erste Spalte zeigt dabei die Hexadezimaladresse des ersten Bytes in der Zeile an – der Opc A9 des 6502 liegt dabei in Byte 0, während das Seitenbyte 4E sich bei beiden Listen in Byte 4 befindet etc. Die zweite Spalte kann ein, zwei oder drei Bytes enthalten. Sie zeigt den Maschinencode an. Die dritte Spalte beginnt mit den mnemotischen Codes der Assemblersprache und stellt die Assembler-Version des Maschinencodes dar. Versuchen Sie nicht, die Befehlsfolge zu verstehen. Für den Augenblick genügt es, daß Sie die Assemblerliste und die Unterschiede zwischen den Versionen des Z80 und des 6502 erkannt haben. Achten Sie dabei auch darauf, daß die Adresse der zweiten Zeile im Maschinencode im lo-hi-Format erscheint, während sie im Assembler das „normale" hi-lo-Format annimmt.

In unserer nächsten Folge werden wir mit der ausführlichen Untersuchung der einzelnen Opcodes anfangen.

Hexadezimalanzeige

In den Mempeek-Programmen läßt sich eine Hexadezimaldarstellung statt der dezimalen Anzeige mit folgenden Änderungen erreichen:

Acorn B

Fügen Sie folgendes hinzu:

```
3000 DEF PROCHXPRINT(DECNUM)
3100 LOCAL X$
3200 X$="0123456789ABCDEF"
3300 HB=INT(DECNUM/16):LB=DECNUM—HB*16
3400 B$=MID$(X$,HB+1,1)+MID$(X$, LB+1,1)+" "
3500 PRINT B$;
3600 ENDPROC
```

und ändern Sie Zeile 600 in:

```
600 PROCHXPRINT(PK%)
```

Spectrum

Fügen Sie folgendes hinzu:

```
  10 LET X$="0123456789ABCDEF"
3000 REM******HEX BYTE S/R*******
3100 LET HB=INT(PK/16):LET LB=PK-HB*16
3200 LET B$=X$(HB+1)+X$(LB+1)+" "
3300 PRINT B$;
3400 RETURN
```

und ändern Sie Zeile 600 in:

```
600 GOSUB 3000
```

Commodore 64

Fügen Sie folgendes hinzu:

```
  10 LET X$="0123456789ABCDEF"
3000 REM******HEX BYTE S/R*******
3100 HB=INT(PK/16):LB=PK-HB*16
3200 B$=MID$(X$,HB+1,1)+MID$(X$,LB+1,1)+" "
3300 PRINT B$;
3400 RETURN
```

und ändern Sie Zeile 600 in:

```
600 GOSUB 3000
```

Die Änderungen veranlassen, daß die Speicherinhalte im Hexadezimalformat angezeigt werden. Die Anfangsadresse und die Anzahl der Bytes müssen jedoch in Dezimalzahlen angegeben werden.

Byte an Byte

In diesem Kursabschnitt werden wir die Entwicklung eines vollständigen Maschinenprogramms von der Definition der Aufgabe über die Umsetzung in die Assemblersprache bis zur Erstellung des Maschinencodes verfolgen.

In der letzten Folge wurde dargestellt, wie BASIC-Programmzeilen bei der Eingabe durch Token und ASCII-Codes ersetzt werden. Das zeigte, in welcher Form sich die „Hochsprache" BASIC am Maschinencode orientiert: Sie besteht aus Folgen einzelner Befehle (die durch den Austausch gegen Token nur eine Ebene über der Maschinensprache liegen) und Befehlsdaten. Da die Schlüsselworte und Daten (Variablen, Zahlen oder Strings) der natürlichen Sprache ähnlich sind und einzelne Befehle deutlich durch Zeilennummern bzw. Doppelpunkte voneinander getrennt werden, erscheint uns BASIC weitaus mehr als Hochsprache als der Interpreter. Daraus folgt, daß auch der Maschinencode nur etwas „kosmetische" Aufbereitung braucht, um für uns verständlich zu werden.

Diese „Kosmetik" ist die Assemblersprache, in der mnemotische Kürzel wie LDA und ADC für die Ein-Byte-Op-Codes eingesetzt werden, die der Microprozessor verarbeiten kann. Außerdem können dabei alphanumerische Symbole wie LABEL1 und TFLAG statt Speicheradressen und numerischer Bezeichnungen verwendet werden. Da der Prozessor die Assemblersprache nicht verstehen kann, muß das Programm vor der Ausführung übersetzt werden. Diese Aufgabe übernimmt entweder ein Programm – der Assembler – oder der Programmierer selbst. Der Vorteil der Assemblersprache liegt darin, daß sie eine direkte Übersetzung des Maschinencodes ist. Durch einfache Umsetzung der mnemotischen Kürzel in Op-codes und der Symbole in Zahlen läßt sich lauffähiger Code herstellen. Da die Assemblersprache leichter zu verstehen ist als der Maschinencode, ist sie bei der Programmentwicklung eine große Hilfe. Normalerweise werden Programme in der Assemblersprache entwickelt und nur im letzten Stadium in den Maschinencode übersetzt. Wir werden beide Sprachen behandeln, um die Grundlagen der Programmierung aufzuzeigen.

Ein Microprozessor kann zwar viele unterschiedliche Vorgänge ausführen, bearbeitet im Grunde genommen aber immer nur die Daten seines Speichers. Dabei greift er entweder direkt auf die RAM- oder ROM-Speicher des Computers zu oder setzt seine eigenen internen Speicher ein, die „Register". Register sind Speicherbytes, die sich direkt im Prozessorchip befinden. Sie haben bestimmte Spezialfunktionen, unterscheiden sich sonst aber nicht von anderen Speicherbytes.

Akkumulator-Register

Das wichtigste Register des Microprozessors ist der Akkumulator. Um ihn einsetzen zu können, müssen wir darin Informationen ablegen – ein Vorgang, den wir „Laden des Akkumulators" nennen. In der Assemblersprache des 6502 führt der Befehl LDA diesen Ladevorgang aus, bei dem Z80 der Befehl LD A. Das Lesen von Informationen aus dem Akkumulator ist ebenso wichtig wie das Laden. Der Assembler des 6502 führt diesen Vorgang mit dem Befehl STA (STore the Accumulator contents) aus. Der Z80 betrachtet Laden und Speichern als einzelne Elemente des gleichen Vorgangs. Das Lesen von Informationen aus dem Akkumulatorregister führt daher ebenfalls der Befehl LD A aus, jedoch mit anderem Format.

Angenommen, Sie möchten ein Assemblerprogramm schreiben, das den Inhalt eines Speicherbytes in das nächste Speicherbyte kopiert: von Byte$09FF in Byte$0A00. In Assembler sieht das folgendermaßen aus:

6502	**Z80**
LDA $09FF	LD A,($09FF)
STA $0A00	LD ($0A00),A

Beachten Sie dabei, daß wir den Inhalt von Byte$09FF in Byte$0A00 kopiert haben, ohne zu wissen, was dieser Inhalt bedeutet. Byte$09FF kann irgendeine Zahl von $00 bis $FF enthalten, unser Programm lädt diese Zahl nur in den Akkumulator und speichert sie vom Akkumulator aus in Byte$0A00. Aus der Assemblerversion des 6502 geht nicht hervor, daß sich LDA auf den Inhalt von Byte$09FF bezieht. Es wird dabei aber eindeutig zwischen dem Ladevorgang (LDA) und dem Speichervorgang (STA) unterschieden. Die Z80-Version hat folgendes Befehlsformat:

OPCODE BESTIMMUNG, (QUELLE)

Dabei sind Speicheradressen in Klammern eingeschlossen, wenn „der Inhalt von" gemeint ist. Mit dieser Methode wird eine äußerst wichtige Unterscheidung zwischen der Adresse eines Bytes und seinem Inhalt vorgenommen.

Da das Assemblerprogramm als Unterrou-

tine eingesetzt werden soll, müssen wir eine Art RETURN integrieren, um die Steuerung an das Ursprungsprogramm zurückzugeben. Der Op-code dieses Befehls für den 6502 ist RTS und der des Z80 RET.

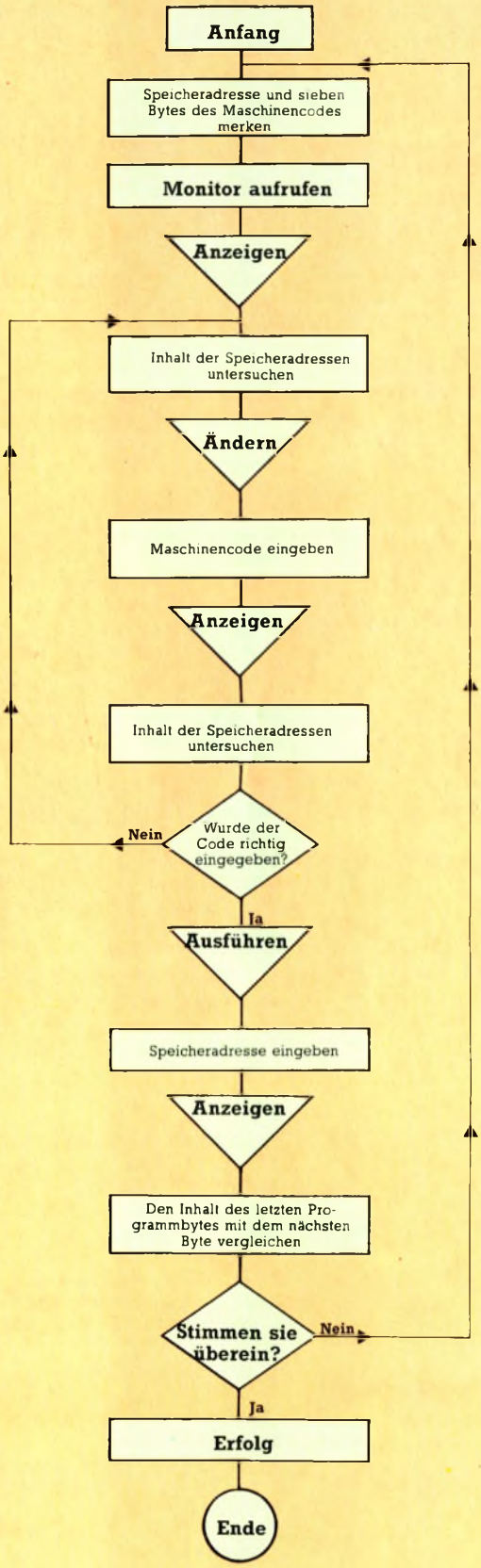
Vor der Ausführung muß die Unterroutine in den Maschinencode übersetzt werden, dann den Code im Speicher unterbringen und schließlich den Microprozessor mit Hilfe des Monitorprogramms veranlassen, das Programm auszuführen. Zuerst müssen wir den Code jedoch übersetzen und entscheiden, an welcher Stelle des Speichers er abgelegt werden soll. Für BASIC-Programmierer ist dies eine ungewöhnliche Überlegung, da sie sich darüber nie Gedanken machen mußten, wo ein BASIC-Programm untergebracht wird – die Entscheidung über den Speicherplatz trafen bereits die Konstrukteure des Systems.

Grundsätzlich kann ein Maschinencodeprogramm an jeder Stelle des Speichers untergebracht und ausgeführt werden. Einige Bereiche eignen sich dafür jedoch besser als andere. Diese Bereiche sind in jeder Maschine verschieden, und daher ergeben sich die folgenden Programmversionen:

6502		
Speicher-adresse	**Maschinen-code**	**Assembler-sprache**
Commodore 64		
$0350	AD 56 03	LDA $0356
$0353	8D 57 03	STA $0357
$0356	60	RTS
Acorn B		
$0070	AD 76 00	LDA $0076
$0073	8D 77 00	STA $0077
$0076	60	RTS
Z80		
16-K-Spectrum		
$7FA0	3A A6 7F	LD A,($7FA6)
$7FA3	32 A7 7F	LD ($7FA7),A
$7FA6	C9	RET
48-K-Spectrum		
$FFA0	3A A6 FF	LD A,($FFA6)
$FFA3	32 A7 FF	LD ($FFA7),A
$FFA6	C9	RET

In jeder Version des Programms wird das letzte Byte des Programms in das darauffolgende Speicherbyte kopiert. So kopiert das Programm z. B. auf dem Spectrum mit 48 K den Inhalt von $FFA6 nach $FFA7. Beachten Sie dabei, daß die Adressen der Assemblersprache das hi-lo Format haben, im Maschinencode jedoch im lo-hi Format erscheinen. Interessant ist auch, daß das mnemotische Kürzel des Z80-Befehls im ersten und zweiten Befehl LD lautet, die Op-codes jedoch verschieden sind: 3A lädt Daten in den Akkumulator, während 32 Daten aus dem Akkumulator liest.

Einsatz des Monitorprogramms



Mit dem auf der nächsten Seite aufgeführten Programm können Sie Speicherinhalte ändern, anzeigen und ausführen lassen. Nach Auswahl einer dieser Möglichkeiten geben Sie die Anfangsadresse im Hexadezimalformat ein, von der an:
1) die Bytes verändert werden sollen;
2) die Byteinhalte angezeigt werden sollen;
3) der Microprozessor das Programm ausführen soll.
Die Eingabe von „X" anstelle einer Hexadezimalzahl läßt das Programm auf die Wahlebene zurückspringen. „Q" beendet das Programm.
Wenn Sie im Änderungsmodus die Adresse angegeben haben, deren Inhalt Sie verändern wollen, wird diese Adresse angezeigt, gefolgt von einem Fragezeichen. Geben Sie jetzt den neuen Inhalt ein.
Nach dem Drücken der RETURN-Taste wird das nächste Speicherbyte in der gleichen Weise angezeigt und kann ebenso geändert werden. Wenn Sie X anstelle einer Adresse eingeben, kehrt das Programm zur Befehlsebene zurück.

In den Speicher laden
Sehen Sie sich das Programm für Ihren Computer an, merken Sie sich die erste Speicheradresse und die sieben Bytes Maschinencode. Mit dem Monitorprogramm können Sie diese sieben Hexzahlen in die sieben Speicherbytes laden, die von der angegebenen Speicheradresse aufwärts liegen:
1) Rufen Sie das Monitorprogramm mit RUN auf. Lassen Sie sich den Inhalt der ersten Adresse ANZEIGEN.
2) Wählen Sie den Befehl ÄNDERN und geben Sie die Speicheradresse und die sieben Bytes des Maschinenprogrammes ein.
3) Wählen Sie wieder ANZEIGEN und überprüfen Sie, ob Sie die Adressen und Bytes des Programms richtig eingegeben haben.
4) Wählen Sie den Befehl AUSFÜHREN und geben Sie die Anfangsadresse an – es scheint nichts zu passieren.
5) Wählen Sie ANZEIGEN und untersuchen Sie die Speicheradressen. Sie werden feststellen, daß der Inhalt des letzten Programmbytes in das darauffolgende Byte kopiert wurde.

Monitorprogramm für den Spectrum

```
48 REM++++++++++++++++++++++
49 REM+                     +
50 REM+  HCAC MONITOR 1     +
51 REM+ ----SPECTRUM------  +
52 REM+ SAVE THIS PROGRAM   +
53 REM+ BEFORE YOU RUN IT   +
54 REM+                     +
55 REM++++++++++++++++++++++
100 GOSUB 1000  :REM *INIT*
200 CLS
300 PRINT " ** HCAC MONITOR 1 CO
MMANDS **"
400 FOR P=1 TO LT:PRINT O$(P):NE
XT P
500 FOR Z=0 TO 1 STEP 0
550 GOSUB 2000  :REM *INPUT*
600 GOSUB (4500+CM*500)
650 NEXT Z
700 STOP
750 REM++++END MAIN PROG++++
1000 REM*****INIT  S/R******
1050 LET LT=4:DIM C$(LT):DIM O$(
LT,24):DIM X$(16)
1100 LET X$="0123456789ABCDEF"
1150 LET C$="ADGQ":LET C1=-48:LE
T C2=10-CODE(C$(1))
1200 LET O$(1)="       A - ALTER
  MEMORY"
1220 LET O$(2)="       D - DISPLA
Y MEMORY"
1240 LET O$(3)="       G - EXECUT
E M/CODE"
1260 LET O$(4)="       Q - EXIT
 PROGRAM"
1300 RETURN
2000 REM*****INPUT S/R******
2100 FOR P=0 TO 1 STEP 0
2150 PRINT:PRINT"COMMAND ??"
2190 IF INKEY$<>"" THEN GO TO 21
90
2200 LET A$=INKEY$:IF A$="" THEN
 GO TO 2200
2250 FOR J=1 TO LT
2300 IF A$=C$(J) THEN LET CM=J:L
ET J=LT:LET P=2
2350 NEXT J:NEXT P:IF A$="Q" THE
N RETURN
2400 PRINT O$(CM)
2450 FOR P=0 TO 1 STEP 0
2500 INPUT"HEX ADDRESS (X=QUIT)"
;A$
2550 GOSUB 5200 :REM CHK&ADJ
2600 NEXT P:IF A$="X" THEN LET C
M=0
2650 RETURN
3000 REM****HEX BYTE S/R****
3010 LET HB=INT(N/16):LET LB=N-H
B*16
3020 LET B$=X$(HB+1)+X$(LB+1)
3030 RETURN
3100 REM*****D-H   S/R******
3110 IF NM<256 THEN LET N=NM:GOS
UB 3000:LET H$="00"+B$:RETURN
3120 LET HI=INT(NM/256):LET LO=N
M-256*HI
3130 LET N=HI:GOSUB 3000:LET H$=
B$
3140 LET N=LO:GOSUB 3000:LET H$=
H$+B$
3150 RETURN
4000 REM*****H-D   S/R******
4050 LET RX=1:LET DN=0:LET HL=LE
N(H$):IF (HL<1) OR (HL>4) THEN L
ET DN=-1:RETURN
4100 FOR H=HL TO 1 STEP -1
4150 LET D$=H$(H)
4200 LET V=CODE(D$)+C1*(D$>="0"
AND D$<="9") + C2*(D$>="A" AND D
$<="F")
4250 IF V>15 THEN LET DN=-1:LET
H=1:NEXT H:RETURN
4300 LET DN=DN+V*RX:LET RX=RX*16
4350 NEXT H:RETURN
4500 REM*****DUMMY S/R*****
4550 RETURN
5000 REM*****ALTER S/R*****
5020 FOR P=0 TO 1 STEP 0
5040 PRINT A$;:INPUT"NEW HEX VAL
UE (X=EXIT) ?";V$
5050 PRINT V$
5060 GOSUB 5340 :REM CHK&OBY
5080 NEXT P:RETURN
5200 REM*CHECK&ADJUST S/R**
5220 IF A$="X" THEN LET P=2:RETU
RN
5240 LET LL=LEN(A$):IF LL>4 THEN
 RETURN
5260 LET H$=A$:GOSUB 4000
5280 IF DN>=0 THEN LET P=2:LET N
M=DN
5300 LET A$=A$+"  ":IF LL<4 THEN
 LET A$="0000"(TO 4-LL)+A$
5320 RETURN
5340 REM**CHECK & OBEY S/R*
5360 IF V$="X" THEN LET P=2:RETU
RN
5380 LET H$=V$:GOSUB 4000
5400 IF (DN<0) OR (DN>255) THEN
RETURN
5420 POKE NM,DN
5440 LET NM=NM+1:IF NM>65535 THE
N LET P=2:RETURN
5460 GOSUB 3100     :REM D-H S/R
5480 LET A$=H$+"  ":RETURN
5500 REM****DISPLAY S/R*******
5520 FOR P=0 TO 1 STEP 0
5540 INPUT"DEC.NO.OF BYTES(X=QUI
T)";N$:IF N$="X" THEN LET P=2:NE
XT P:RETURN
5560 LET BN=VAL(N$):IF (BN>0) AN
D (BN+NM<65536) THEN LET P=2
5580 NEXT P
5600 FOR B=NM TO (NM+BN-1) STEP
4
5620 LET L$="":LET NM=B:GOSUB 31
00
5640 PRINT H$;TAB(6);
5660 FOR C=0 TO 3
5680 LET N=PEEK(B+C):LET K$="."
5700 GOSUB 3000 :REM D-H S/R
5720 PRINT TAB(6+4*C);B$;
5740 IF N=0 THEN LET K$="■"
5760 IF (N>31) AND (N<128) THEN
LET K$=CHR$(N)
5780 LET L$=L$+K$
5800 NEXT C
5820 PRINT TAB(26);L$
5840 NEXT B:RETURN
6000 REM*****EXECUTE S/R****
6050 RANDOMIZE USR(NM):RETURN
6500 REM*****EXIT  S/R******
6550 PRINT TAB(5);"■■■■END OF PR
OGRAM■■■■"
6600 LET Z=2:RETURN
```

Acorn B

```
30 REM************************
40 REM*    HCAC MONITOR 1     *
41 REM* ---------BBC---------- *
42 REM*  CHANGE THE SPECTRUM   *
43 REM*  VERSION AS FOLLOWS:   *
44 REM*                       *
45 REM*  REPLACE CODE( BY ASC( *
47 REM*                       *
50 REM*  ADD,CHANGE,OR DELETE  *
51 REM*       AS DIRECTED:    *
52 REM*                       *
53 REM************************
60 *TV 255
70 MODE 7
200 PRINT CHR$(147);CHR$(142)
600 ON CM GOSUB 5000,5500,6000,6500
1050 LT=4:DIM C$(LT),O$(LT)
1150 C$(1)="A":C$(2)="D":C$(3)="G":C$(4
)="Q":C1=48:C2=ASC(C$(1))-10
2190 ----------DELETE----------
3020 B$=MID$(X$,HB+1,1)+MID$(X$,LB+1,1)
4150 D$=MID$(H$,H,1)
4500 ----------DELETE----------
4550 ----------DELETE----------
5050 ----------DELETE----------
5300 A$=A$+"  ":IF LL<4 THEN A$=LEFT$("0
000",4-LL)+A$
5420 ?(NM)=DN
5680 N=?(B+C):K$="."
5740 IF N=12 THEN K$=CHR$(255)
6050 CALL NM:RETURN
6600 Z=1:RETURN
```

Commodore 64

```
49 REM************************
50 REM*    HCAC MONITOR 1     *
51 REM* ---------64M---------- *
52 REM*  CHANGE THE SPECTRUM   *
53 REM*  VERSION AS FOLLOWS:   *
54 REM*                       *
55 REM*  REPLACE ALL INSTANCES *
56 REM*  OF:"LET P=2" BY "P=1" *
57 REM*                       *
58 REM*  REPLACE CODE( BY ASC( *
59 REM*                       *
60 REM*  AND CHANGE OP DELETE  *
61 REM*       AS DIRECTED:    *
62 REM*                       *
63 REM************************
200 PRINT CHR$(147);CHR$(142)
600 ON CM GOSUB 5000,5500,6000,6500
1050 LT=4:DIM C$(LT),O$(LT)
1150 C$(1)="A":C$(2)="D":C$(3)="G":C$(4
)="Q":C1=48:C2=ASC(C$(1))-10
2190 ----------DELETE----------
3020 B$=MID$(X$,HB+1,1)+MID$(X$,LB+1,1)
4150 D$=MID$(H$,H,1)
4500 ----------DELETE----------
4550 ----------DELETE----------
5050 ----------DELETE----------
5300 A$=A$+"  ":IF LL<4 THEN A$=LEFT$("0
000",4-LL)+A$
5740 IF N=0 THEN K$=CHR$(122)
6050 SYS(NM):RETURN
6600 Z=1:RETURN
```

Mit diesem Programm können Sie sich den Inhalt des Speichers anzeigen lassen, Speicherbytes ändern und ein gespeichertes Maschinencodeprogramm ausführen lassen.

Register und Speicher

Hier wird untersucht, wie im Speicher ein Platz für Maschinencodeprogramme gefunden werden kann. Weiterhin sehen wir, wie die Zentraleinheit den Inhalt des Akkumulators behandelt.

In unserer letzten Folge haben wir ein einfaches Assemblerprogramm entwickelt, es in den Maschinencode übersetzt (assembliert), in den Speicher geladen und ausgeführt. Die beiden letzten Aufgaben wurden dabei von einem Monitorprogramm übernommen. Ein höher entwickeltes Programm mit integriertem Assembler hätte auch die Assemblierung ausführen können. Wenn Sie erst die Grundlagen des Vorgangs verstanden haben und Ihre Programme länger werden, ist die Handübersetzung eine lästige und fehleranfällige Aufgabe; zu dem Zeitpunkt sollten Sie sich ein Assemblerprogramm für Ihre Maschine anschaffen.

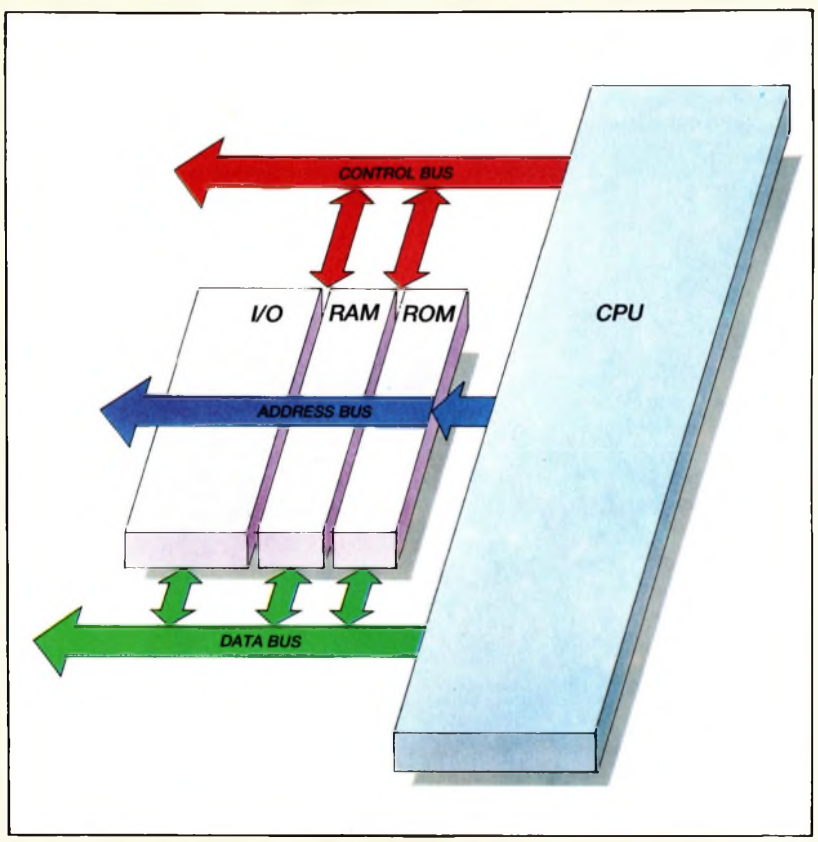
Unser kurzes Programm im Maschinencode enthält einige wesentliche Merkmale: Mit einem CPU-Register haben wir Daten verschoben, weiterhin brauchten wir eine Entscheidung, in welchem Bereich des Speichers wir den Code unterbringen wollen, und außerdem mußte der Microprozessor veranlaßt werden, den Code auszuführen. All diese Aspekte der Assemblerprogrammierung können einen Anfänger verwirren. Wir werden daher genau darauf eingehen und uns zunächst mit dem für Maschinencode reservierten Speicherplatz beschäftigen.

Speicherbereiche für Maschinensprache

Zwar wäre es möglich, überhaupt kein BASIC einzusetzen und damit den gesamten dafür reservierten Bereich zur Verfügung zu haben, doch normalerweise wird zumindest ein Teil des Programms in BASIC geschrieben und der Maschinencode nur für Zwecke eingesetzt, für die BASIC zu langsam ist (z. B. Bewegungsabläufe auf dem Bildschirm). Wenn BASIC und Maschinenprogramme aber gleichzeitig im RAM vorhanden sein sollen, dann müssen wir dem BASIC einige Bereiche wegnehmen und dem Maschinencode zur Verfügung stellen. Wir können dafür die Grenzen des Textbereiches für BASIC-Programme verändern oder vom BASIC oder Betriebssystem nicht genutzte Teile belegen. Auf dem Acorn B lassen sich die Grenzen des BASIC leicht verschieben, da die entsprechenden Adressen in den Systemvariablen PAGE, TOP, LOMEM und HIMEM enthalten sind. So zeigt PAGE z. B. auf den Anfang des Bereiches für BASIC-Programmtexte – normalerweise $1200. Wenn Sie folgenden Befehl eingeben:

```
PAGE=PAGE+500
```

speichert das Betriebssystem alle BASIC-Programme 500 Bytes höher im Speicher und stellt



Der Aufbau kleiner Systeme

In seiner einfachsten Form besteht ein typisches Computersystem aus einem Speicher und einer CPU. Der Speicher enthält ROM-Chips mit Systemprogrammen, RAM-Chips und Spezialchips für die Steuerung von Ein- und Ausgabevorgängen.
Busse tragen Daten und Steuersignale von einem Systemelement zum anderen. Sie sind Flachkabeln ähnlich und können ein Byte oder auch mehr Daten gleichzeitig übertragen. Einige Busse können Daten nur in eine Richtung übermitteln, z. B. *Adreßbusse*, während andere in beiden Richtungen funktionieren, wie z. B. *Datenbusse*. Ein *Steuerbus* überträgt die internen Schaltimpulse des Systems und öffnet und schließt damit Gatter, die den Datenfluß steuern.

Ihnen damit einen Bereich von 500 Bytes für Maschinencodeprogramme zur Verfügung. Bei anderen Systemen kann die gleiche Wirkung erzielt werden, indem die Systemzeiger (siehe Speicherdiagramme) über POKE mit höheren Adressen belegt werden. Sie können auch die obere Grenze des Bereiches für BASIC-Programmtexte nach unten verlegen. Auf dem Spectrum erzielt der Befehl CLEAR gefolgt von einer Adresse diese Wirkung. Bei all diesen Grenzverschiebungen muß nur darauf geachtet werden, daß noch genügend Platz für die BASIC-Programme übrigbleibt.

Auch im für das Betriebssystem reservierten RAM lassen sich kleine Blöcke mit freiem Speicherplatz finden. Ein typisches Beispiel dafür ist der Cassettenbuffer des Commodore 64. Er besteht aus 192 RAM-Bytes (von $033C bis $03FB) und wird von dem Betriebssystem nur beim Einsatz eines Cassettenrecorders angesprochen. Dieser Platz reicht vielen Programmierern völlig aus.

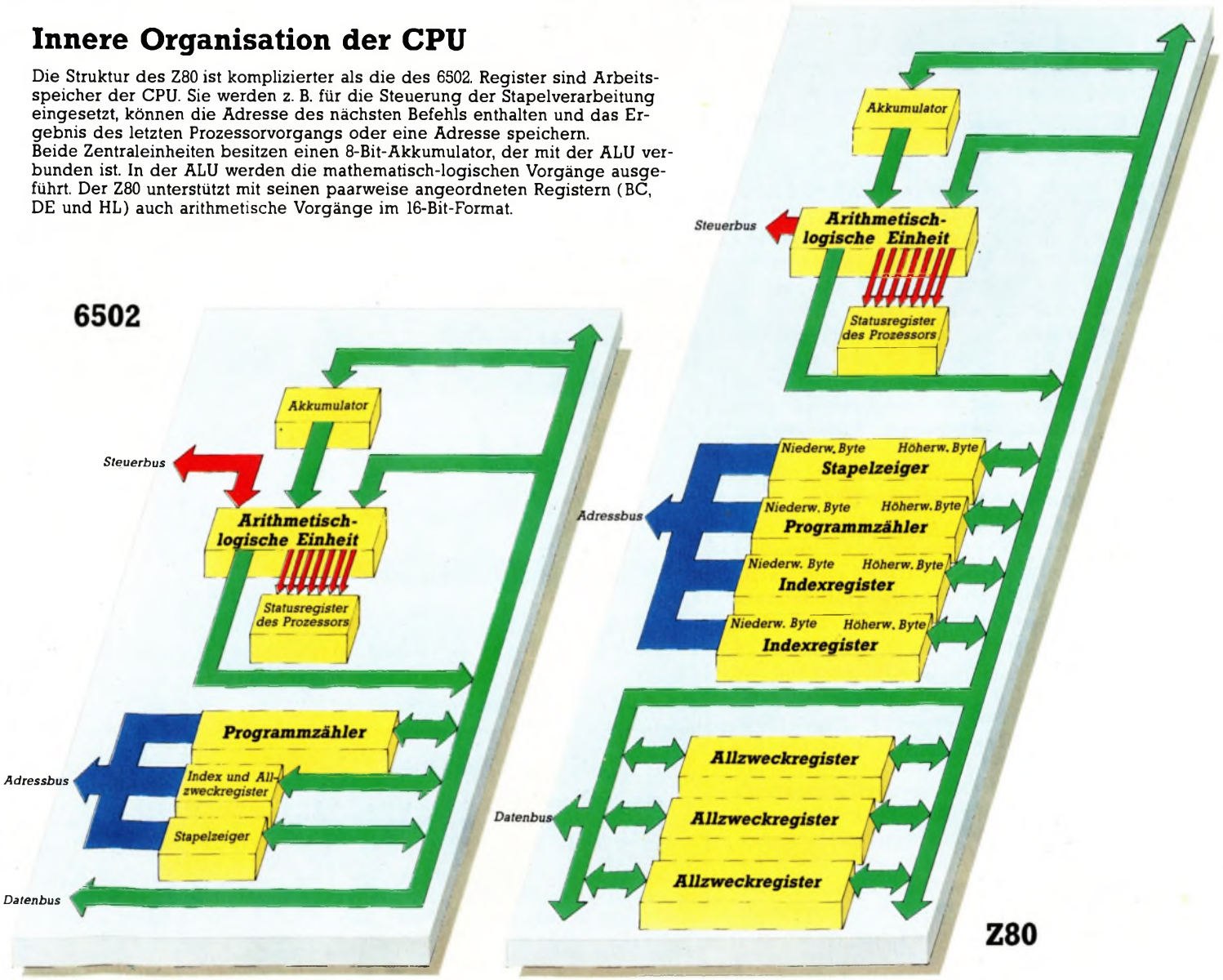
Doch selbst in BASIC-Programmen lassen sich kleinere freie Speicherbereiche finden, z. B. in REM-Zeilen. Der Speicherbereich für BASIC-Text enthält in der Zeile

```
10 REM**************************
```

25 Bytes mit der Zahl $2A (ASCII-Code für "*"). Diese Bytes werden vom Betriebssystem oder vom Interpreter nie angesprochen, da sie den Befehl REM als „ignoriere den Rest dieser Zeile" interpretieren. In einer derartigen Programmzeile können die Sterne ungestört vom Maschinencode überschrieben werden. Diese scheinbar unsaubere Methode (oft für Programme auf dem erweiterten ZX81 eingesetzt) hat einen großen Vorteil: Wenn Sie den BASIC-Programmtext mit SAVE speichern und dann mit LOAD wieder aufrufen, bleibt der Maschinencode erhalten. Bei anderen Methoden muß der Maschinencode normalerweise getrennt

Innere Organisation der CPU

Die Struktur des Z80 ist komplizierter als die des 6502. Register sind Arbeitsspeicher der CPU. Sie werden z. B. für die Steuerung der Stapelverarbeitung eingesetzt, können die Adresse des nächsten Befehls enthalten und das Ergebnis des letzten Prozessorvorgangs oder eine Adresse speichern.
Beide Zentraleinheiten besitzen einen 8-Bit-Akkumulator, der mit der ALU verbunden ist. In der ALU werden die mathematisch-logischen Vorgänge ausgeführt. Der Z80 unterstützt mit seinen paarweise angeordneten Registern (BC, DE und HL) auch arithmetische Vorgänge im 16-Bit-Format.



Akkumulator
Der Akkumulator ist das zentrale Register der CPU. Alle arithmetisch-logischen Vorgänge und auch die Datenübertragung werden über dieses Register geleitet – der 6502 macht davon jedoch mehr Gebrauch als der Z80.

Arithmetisch-logische Einheit
Sie enthält ein Additionsmodul für Binärzahlen und logische Gatter, durch die der Zugang zu den individuellen Bits der Register und des Datenbusses möglich wird. Mit der entsprechenden Steuerung können damit Addition, Subtraktion und Boolesche Vergleiche ausgeführt werden.

Statusregister des Prozessors
Bei fast jedem Vorgang in der CPU erhalten die einzelnen Bits dieses Registers Informationen über Teilaspekte des Vorgangs, z. B. ein Übertragungsbit aus dem Ergebnis einer Addition.

Programmzähler
Dieses Register enthält die Speicheradresse mit dem nächsten Op-Code, den die CPU ausführen soll. Die BASIC-Funktion USR (Adresse) bewirkt, daß die angegebene Adresse direkt in den Programmzähler geladen wird und die CPU den von dieser Adresse an gespeicherten Code aufrufen und ausführen kann.

Stapelzeiger
In diesem Register ist die Adresse des nächsten freien Bytes des für die CPU reservierten RAM-Bereiches gespeichert. Wenn die CPU Daten auf dieser Adresse ablegt, wird der Stapelzeiger automatisch auf das nächste freie Byte gesetzt.

Index- und Allzweckregister
Indexregister werden von der CPU eingesetzt, um den Speicher auf mehrere verschiedene Arten ansprechen zu können. Allzweckregister werden als allgemeine Ablage oder für Spezialaufgaben der CPU eingesetzt.

gespeichert werden. Dennoch hat die REM-Methode einen großen Nachteil: Beim Anzeigen eines Programmtextes mit LIST interpretiert das Betriebssystem den Maschinencode als ASCII-Zeichen und kann damit Chaos auf dem Bildschirm hervorrufen. Hier wird auch deutlich, warum in den Beispielprogrammen des Acorn B und des Spectrum Warnungen enthalten sind. Der Commodore 64 lädt seinen Maschinencode in den Cassettenbuffer, der Acorn und der Spectrum jedoch in die erste REM-Zeile – daher die Warnung, diese Programmversionen nicht mit LIST anzeigen zu lassen.

Ausführung starten

Wenn Sie ein Assemblerprogramm geschrieben, in den Maschinencode übersetzt und in den RAM-Bereich Ihrer Wahl geladen haben, können Sie es ausführen lassen. Dieser Vorgang wird mit den BASIC-Befehlen CALL (nur Acorn), SYS (nur Commodore 64) oder USR (alle drei Maschinen) in Gang gesetzt. Allen drei Befehlen folgt die Adresse des ersten Bytes im Maschinencode. Für den Interpreter bedeutet das: „Führe den Maschinencode aus, der an der angegebenen Adresse anfängt, und gehe bei einem Rücksprungbefehl des Maschinencodes (RET oder RTS) auf den nächsten Befehl des BASIC-Programmes." Der Ablauf ähnelt dem BASIC-Befehl GOSUB.

In unserer letzten Folge haben wir ein Programm geschrieben, das den Inhalt eines Bytes in ein anderes kopiert. Wir haben dazu den Akkumulator mit dem Inhalt der ersten Adresse geladen und dann den Inhalt des Akkumulators in die andere Adresse übertragen. Dieser Vorgang zeigt die zentrale Rolle der CPU: Alle Daten und die Steuerung müssen vom Speicher in die CPU und von dort wieder in den Speicher fließen. Während im BASIC LET X=Y (d.h. „kopiere den Inhalt von Y direkt nach X") möglich ist, können wir in der Assemblersprache nur vom Speicher in die CPU und von der CPU in den Speicher kopieren. Die Register der CPU (siehe Bild) sind RAM-Bytes, die sich im Prozessor selbst befinden. In diesen Registern werden Daten des Speichers abgelegt oder bearbeitet. Sowohl der Z80 als auch der 6502 besitzen ein Register, das Akkumulator heißt und von den meisten Befehlen der Assemblersprache angesprochen wird. In ihm finden fast alle Rechenoperationen statt.

Wenn Sie z. B. die Zahlen $42 und $07 addieren wollen (das $-Symbol kennzeichnet die Ziffern als Hexadezimalzahlen), brauchen Sie nur eine dieser Zahlen in den Akkumulator zu laden und die andere darauf zu addieren – die Summe wird sich in diesem Register regelrecht „ansammeln" (englisch: accumulate). Im folgenden Kasten finden Sie die Befehle des Z80- und des 6502-Microprozessors für diesen Vorgang.

Z80	6502
LD A,$42 ADC A,$07	LDA #$42 ADC #$07

Beide Befehle des Z80 geben die Zahlen direkt an, während den Zahlen der Version des 6502 ein # vorangestellt werden muß, damit sie als Zahlen und nicht als Adressen interpretiert werden. LDA #$65 bedeutet daher: „lade die Zahl $65 in den Akkumulator", während LDA $65 die Bedeutung hat: „lade den Inhalt des Bytes mit der Adresse $65 in den Akkumulator". Auf die gleiche Weise funktioniert der Additionsbefehl (das mnemotische Kürzel ist in beiden Versionen zufällig identisch). ADC bedeutet: „addiere einen festgelegten Operanden zu dem Inhalt des Akkumulators." Die Zahlen $42 und $07 sind in unserem Beispiel sogenannte „Konstante", so daß LDA #$42 als „lade den Akkumulator mit der Konstanten #$42" verstanden wird.

Nach der Ausführung der Befehle steht die Summe beider Zahlen im Akkumulator. Da Sie dies nicht „sehen" können, muß der Inhalt des Akkumulators in ein RAM-Byte übertragen werden, das sich abfragen läßt. Das vollständige Programm sieht folgendermaßen aus:

Maschinencode	Assemblersprache
Z80	

A7
3E 42
CE 07
32 ?? ??
C9
	AND A
	LD A,$42
	ADC A,$07
	LD BYTE1,A
	RET

6502	
18 A9 42 69 07 8D ?? ?? 60	CLC LDA #$42 ADC #$07 STA BYTE1 RTS

Wir werden für den Augenblick nicht auf die Bedeutung des ersten Befehls beider Programme eingehen. Der vierte Befehl enthält statt einer direkten Adresse das Symbol BYTE1. Da der Wert von BYTE1 von Maschine zu Maschine variiert, haben wir lediglich ein Symbol eingesetzt.

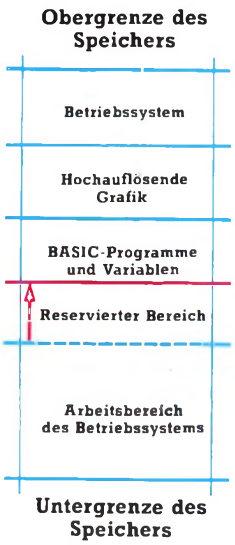
Jetzt müssen Sie sich überlegen, wo Sie den Maschinencode abspeichern wollen und welche Adresse BYTE1 zugeordnet werden soll. Wählen Sie einen Speicherbereich für Ihren Code aus und setzen Sie an die Stelle von BYTE1 die Adresse des Bytes, das dem Programm unmittelbar folgt. Diese Adresse muß im Maschinencode das lo-hi-Format haben. Danach laden, starten und untersuchen Sie den Maschinencode mit dem Monitorprogramm. Der Inhalt des Ergebnisbytes sollte $49 sein.

Assemblerbefehle

Maschinencodeprogramme, die im Assemblerformat geschrieben werden, müssen am Anfang bestimmte Anweisungen enthalten. In dieser Folge wird untersucht, welche Funktionen einige dieser „Assembleranweisungen" ausführen.

Acorn B

Direkte Eingabe des Maschinencodes:
PRINT. ~ PAGE
gibt die Anfangsadresse des reservierten Speicherbereiches in Hex an. Geben Sie dann ein:
PAGE=PAGE+N
wobei N die Anzahl (Dezimalzahl) der Bytes ist, die Sie für Ihren Maschinencode reservieren wollen.



In der letzten Folge haben wir ein einfaches Maschinencodeprogramm geschrieben, das zwei Zahlen im Akkumulator addiert und im Speicher ablegt. Obwohl das Programm sehr einfach war, wurden dadurch viele grundlegende Abläufe der Maschinencodeprogrammierung deutlich. Sehen Sie sich dieses Programm nochmals mit den zugehörigen Adressen an. Darin wurden der Code von der Speicherstelle $0000 an geladen und das Resultat auf die Speicheradresse $0009 gelegt. (Dies ist jedoch nur für Anschauungszwecke gedacht – jeder Versuch, diese Speicheradressen einzusetzen, würde den sofortigen Absturz des Systems verursachen.) Hier die Programmversionen:

Speicheradresse	Maschinencode	Assemblerbefehle
	Z80	
0000	A7	AND A
0001	3E 42	LD A,$42
0003	CE 07	ADC A,$07
0005	32 09 00	LD BYTE1,A
0008	C9	RET
	6502	
0000	18	CLC
0001	A9 42	LDA #$42
0003	69 07	ADC #$07
0005	8D 09 00	STA BYTE1
0008	60	RTS

Der vierte Befehl (der den Inhalt des Akkumulators in $0009 speichert) enthält im Assemblercode beider Programme keine Hexadezimaladresse, sondern ein Symbol: BYTE1. In der Maschinencodeversion dieses Befehls finden wir jedoch den Op-code für "Lade die folgende Speicherstelle aus dem Akkumulator", gefolgt von 09 00, den zwei Bytes der Adresse $0009 im lo-hi-Format.

Hier zeigt sich ein interessanter Mechanismus der Übersetzung des in Assembler geschriebenen Programms in den Maschinencode. Ebenso wie zur Vereinfachung der Programme die Hexzahlen der Befehle durch mnemotische Kürzel ersetzt wurden (beispielsweise 8D und C9 durch STA und RET), können schwer lesbare Adressen wie $0009 durch Symbole wie BYTE1 ersetzt werden. Dieser Vorgang ist der gleiche wie die Initialisierung von Variablen in BASIC. Auch die Gründe sind die gleichen: Programme werden übersichtlicher, die Möglichkeit, Fehler einzugeben, verringert sich, und der Code ist leichter zu bearbeiten. Wenn beispielsweise im gesamten Programm eine Adresse geändert werden soll, genügt es, dem entsprechenden Symbol anfänglich einen anderen Wert zuzuordnen, den das gesamte Programm ohne weiteres Editieren verwendet.

In der BASIC-Programmierung ist dieser Vorgang leicht zu verstehen. Wie aber sieht der Assemblerbefehl für den BASIC-Befehl LET BYTE1=$0009 aus? Wenn zur Übersetzung in den Maschinencode kein Assemblerprogramm eingesetzt wird, müssen wir diese Arbeit selbst ausführen. Bei Verwendung des Assemblers aber brauchen wir nur den entsprechenden Befehl an den Anfang unseres Codes zu setzen. Hier die Version für den Z80:

Z80		
0000		BYTE1 EQU $0009
0000	A7	AND A
0001	3E 42	LD A,$42
0003	CE 07	ADC A,$07
0005	32 09 00	LD BYTE1,A
0008	C9	RET

BYTE1 erhält dabei eine eigene Spalte, die Symbol-(oder Label-)feld genannt wird. Im Op-codefeld steht ein neues mnemotisches Kürzel (EQU bedeutet "Weise den Wert ... zu"), während das Operandenfeld den Wert enthält, der BYTE1 zugeordnet wird (in diesem Fall $0009).

Beachten Sie, daß EQU zwar im Op-codefeld eingetragen ist und wie ein Maschinencodebefehl aussieht, aber dennoch weder Sprachelement des Assemblers noch des Maschinencodes ist. Ein derartiges mnemotisches Kürzel wird „Pseudo-Befehl" genannt. EQU teilt dem Assembler mit, daß er bei der Übersetzung in den Maschinencode das vorstehende alphanumerische Symbol (hier: BYTE1) im gesamten Programmtext durch den nachstehenden Wert (hier: $0009) ersetzen soll. Wie Sie bereits wissen, wird beim Einsatz eines Assemblerprogramms der Code in der Assemblersprache geschrieben (entweder zur Speicherung auf Cassette oder Diskette oder direkt in den Arbeitsspeicher) und dann vom Assembler in die Maschinensprache übersetzt.

Der Ausdruck eines Assemblerprogramms besteht normalerweise aus Assemblerbefeh-

len wie oben aufgeführt und ihrer Entsprechung im Maschinencode in Form von Hex-Bytes. Der Maschinencode kann dabei für den späteren Einsatz gespeichert werden oder zur direkten Ausführung vorgesehen sein. Ein Assembler kann aber auch andere Aufgaben ausführen, z. B. die Eingabe von Programmadressen. Diese Aufgabe übernimmt der Pseudobefehl ORG. Das Programm sieht dann folgendermaßen aus:
Interessant ist hierbei, daß die Adresse der er-

Z80		
0000		ORG $A000
A000		BYTE1 EQU $0009
		AND A
A001	3E 42	LD A,$42
A003	CE 07	ADC A,$07
A005	32 09 00	LD BYTE1,A
A008	C9	RET

sten Zeile $0000 ist, die der darauf folgenden Zeile jedoch $A000. Hierin zeigt sich die Auswirkung des ORG-Befehls. In den Op-codefeldern der Pseudobefehle stehen keine Hex-Bytes, da diese nicht Teil des Programms sind und daher auch nicht in den Maschinencode übersetzt werden. Pseudobefehle sind von Assembler zu Assembler verschieden. EQU wird beispielsweise manchmal durch "=" dargestellt und ORG durch ".=". Die Wirkung ist jedoch die gleiche, und wir werden ORG und EQU weiterhin verwenden.

Mnemotische Kürzel

Es mag Ihnen vielleicht aufgefallen sein, daß wir von Anfang an schon einen Pseudobefehl eingesetzt haben: "$" – die Markierung der Hexadezimalzahlen. Dieses Zeichen ist eine Anweisung an den Assembler, den folgenden Wert als Hexzahl anzusehen. Auch das Zeichen "#", das wir in der letzten Folge vorgestellt hatten, kennzeichnet eine „Konstante". Es legt fest, daß der darauf folgende Wert eine direkte Zahlenangabe ist und kein Zeiger oder Symbol. Genauer betrachtet ließe sich die gesamte Assemblersprache als eine Folge von Pseudobefehlen ansehen. Und Sie können ihre eigenen mnemotischen Kürzel erfinden – vorausgesetzt natürlich, sie entsprechen unmittelbar dem Befehlssatz des Maschinencodes.

In diesem Kurs verwenden wir natürlich auch weiterhin die vorgegebenen mnemotischen Kürzel. Von Zeit zu Zeit sollten wir uns aber daran erinnern, daß alles, was als Maschinencode bezeichnet wird, nur Symbolcharakter hat. Die CPU verarbeitet Stromimpulse, die über ihre Kontakte geleitet werden. Wie diese Impulsfolgen dargestellt werden, ist nur eine Sache der Übereinkunft.

Nun zu einem anderen Gebiet: den unterschiedlichen Übersetzungen von LDA und LD A. Sehen Sie sich folgende Zeilen an:

AD ?? ??	LDA $????	(6502)
3A ?? ??	LD A, ($????)	(Z80)

Sie bedeuten beide: "Lade den Akkumulator mit dem Byte der Adresse ????". Der Op-code für diesen Ladevorgang ist $AD (für den 6502) und $3A (für den Z80). Wenn Sie diese Codes mit dem Befehl "Lade den Akkumulator mit der Konstanten $42" unseres Programms vergleichen, dann sehen Sie aber die Op-codes $A9 (6502) und $3E (Z80). Die Unterschiede lassen sich leicht erklären: Obwohl in beiden Fällen die gleiche Befehlsart ausgeführt wird (Daten werden in den Akkumulator geladen), kommen die Daten aus unterschiedlichen Quellen. Die CPU muß daher auch unterschiedliche Op-codes erhalten.

Im ersten Fall werden die Daten aus einem Speicherbyte geladen, dessen Adresse angegeben ist. Über den Inhalt des Bytes ist dabei nichts bekannt. Die CPU erhält nur die Anweisung, das Byte in den Akkumulator zu kopieren. Die drei Bytes des Maschinencodes AD ?? ?? und 3A ?? ?? befehlen der CPU "Interpretiere die zwei Bytes, die diesem Op-code folgen, als die absolute Adresse der Datenquelle".

Im zweiten Fall bestehen die Daten, die in den Akkumulator geladen werden sollen, aber aus dem Byte, das dem Op-code unmittelbar folgt. Die Maschinencodebytes A9 42 und 3E 42 teilen der CPU mit: "Interpretiere das Byte, das dem Op-code folgt, als Datenquelle". Die Op-codes A9 oder 3E enthalten daher den Befehl, das auf den Op-code folgende Byte in den Akkumulator zu laden. Da der Programmzähler immer die Adresse des nächsten Befehls enthält, kann die CPU die Adresse des Quellenbytes berechnen und dann das Byte der berechneten Adresse laden.

Diese beiden Beispiele zeigen, wie einfach und unkompliziert die Vorgänge innerhalb der CPU sind. Eine Befehlsklasse (etwa ein Fünftel des gesamten Instruktionssatzes) besteht nur aus Kopiervorgängen, die die Daten eines adressierten Bytes in eines der internen Register laden. All diese „primitiven" Kopiervorgänge unterscheiden sich nur durch das Format, das die Adresse des Quellenbytes darstellt.

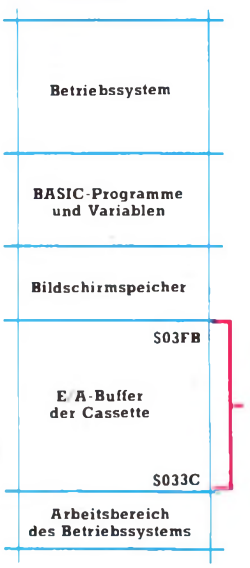
Es mag zunächst vielleicht etwas verwirrend sein, wenn wir derart ausführlich auf die Vorgänge im Inneren der CPU eingehen. Wenn Sie nur den Assemblercode eines bestimmten Prozessors lernen wollen, genügt es, sich den Befehlssatz und ein paar Tricks anzueignen, und Sie können mit der Programmierung anfangen. Sind Sie jedoch mit den Grundlagen vertraut, dann haben Sie nicht nur eine weitere Programmiersprache erlernt, sondern können sich leichter in die Abläufe anderer Prozessoren und Assemblerprogramme einfinden. Dadurch wird die Programmierarbeit leichter, und die Codes laufen schneller.

Spectrum

Direkte Eingabe des Maschinencodes:
LET RTOP=PEEK(23730)
+256*PEEK(23731)
LET RTOP=RTOP-N
PRINT "RTOP= ";RTOP
N ist die Anzahl der Bytes, die Sie für ihren Maschinencode reservieren wollen. Der reservierte Bereich beginnt bei 1+RTOP.

Commodore

Verwenden Sie den Cassettenbuffer von $033C bis $03FB zur Speicherung des Maschinencodes.



Achtung!
Diese Speicherzeiger müssen unmittelbar nach dem Anschalten der Maschine umgestellt werden. Es darf sich noch kein BASIC-Programm im Speicher befinden.

Assembler-übung

1) In dem nebenstehenden Kasten haben wir ein einfaches Programm im Assemblerformat abgedruckt. Assemblieren Sie das Programm in den Maschinencode und bestimmen Sie die Speicheradressen.
2) Welche Befehle fehlen in dem Programm?
3) Welche Auswirkungen hat das Programm auf die Register und die entsprechenden RAM-Bytes?
4) Was bedeutet der Begriff „Konstante"? Welche Arten von Daten gibt es noch?
5) Wenn BYTE1 des Programms als Adresse angesehen würde, welche Speicherseite würde das betreffen?
Die in den Programmen angegebenen Werte haben nur Übungscharakter und sind nicht für einen speziellen Computertyp gedacht.

Speicheradresse	Maschinencode	Assemblerformat		
		6502		
		START	EQU	$A000
		BYTE1	EQU	$45
		BYTE2	EQU	$38
			ORG	START
			LDA	#BYTE1
			CLC	
			ADC	#BYTE1
			STA	BYTE1
			ADC	#BYTE2
			STA	BYTE2
		Z80		
		START	EQU	$A000
		BYTE1	EQU	$45
		BYTE2	EQU	$38
			ORG	START
			LD	A,BYTE1
			AND	A
			ADC	A,BYTE1
			LD	(BYTE1),A
			ADC	A,BYTE2
			LD	(BYTE2),A

Die Auswirkungen von Maschinencodebefehlen

Befehl 1: Lade den Akkumulator mit dem Inhalt der Adresse 1.
Z80 : LD A, (ADDR1)
6502 : LDA ADDR1

Befehl 2: Addiere dazu den Inhalt der Adresse 2.
Z80 : ADC A, (ADDR2)
6502 : ADC ADDR2

Befehl 3: Speichere den Inhalt des Akkumulators in die Adresse 2.
Z80 : LD (ADDR2),A
6502 : STA ADDR2

Mit Ladeoperationen wie LDA ADDR1 oder LD(ADDR2),A wird immer der Inhalt der Quellenadresse in die Bestimmungsadresse kopiert. Der bisherige Inhalt der Bestimmungsadresse wird dabei überschrieben, während sich der Inhalt der Quellenadresse durch den Kopiervorgang nicht verändert.

Flagge zeigen

In diesem Artikel wird untersucht, wie mathematische Abläufe von der Systemarchitektur gesteuert werden. Außerdem sehen wir uns die Rolle der Statusregister bei der Addition genauer an.

In der Assemblersprache des Z80 und des 6502 ist ADC ein wichtiger Befehl. Das mnemotische Kürzel bedeutet „Addiere mit Übertrag" (oder „Add with Carry") und stellt eine der Hauptinstruktionen der Assemblerprogrammierung dar. Das Übertragsbit („carry bit") erfüllt dabei eine wesentliche Funktion. Sehen wir uns die Addition zweier Zahlen im Akkumulator genauer an:

```
  A7      =      10100111
+ 3E       =    + 00111110
-------------------------
  E5      =      11100101
```

Da der Akkumulator ein Acht-Bit-Register ist, dürfen die geladenen Zahlen und die Ergebnissumme nur Zahlen von je 8 Bit zwischen $00 und $FF sein. Das bedeutet jedoch nicht, daß Additionsvorgänge nur Ergebnisse unter $100 erzeugen dürfen. Die folgende Addition überschreitet die Grenze des Akkumulators:

```
  FF      =      11111111
+ FF       =    + 11111111
-------------------------
 1FE      =     111111110
```

Die Addition der beiden höchsten Zahlen, die ein Byte darstellen kann, ergibt eine Summe, die nur ein Neun-Bit-Akkumulator fassen könnte. Mit einem zusätzlichen Bit ließe sich also die größte Zahl, die im Akkumulator entstehen kann, problemlos anzeigen. Dieses zusätzliche Bit wird nur benötigt, wenn ein Übertrag vom höchstwertigen Bit des Akkumulators entsteht.

Das Extrabit wird Übertragsbit genannt. Es befindet sich in einem Acht-Bit-Register, das als „prozessor status register" (PSR) bekannt ist. Dieses ist mit dem Akkumulator und der ALU so verbunden, daß die einzelnen Bits des PSR nach jeder Akkumulatoraktivität je nach Ergebnis gesetzt oder gelöscht werden. Der Inhalt der Statusregister kann zwar als einfache Zahl angesehen werden, wird gewöhnlich aber als eine Anordnung von acht binären Flags verstanden, deren individueller Zustand die Auswirkungen des letzten Vorgangs im Akkumulator anzeigt. (Ein Flag ist eine Variable, deren Inhalt den Zustand oder Wahrheitswert einer Bedingung wiedergibt und nicht als absoluter Zahlenwert angesehen wird. Ein Flag zeigt normalerweise nur zwei Zustände an: gesetzt oder nicht gesetzt; an oder aus; 0 oder 1.)

Wenn im Akkumulator ein Vorgang stattfindet, der vom achten Bit aus einen Übertrag erzeugt, dann wird das Übertragsflag des PSR automatisch auf 1 gesetzt. Ein Vorgang, der keinen Übertrag erzeugt, setzt das Flag zurück auf 0. Das Übertragsflag wird jedoch nur angesprochen, wenn der Befehl einen Übertrag ermöglicht. Abläufe, bei denen der Akkumulator als Zwischenspeicher eines Lade- oder Speichervorgangs eingesetzt wird, wirken sich auf das Übertragsflag nicht aus.

Addition und Übertrag

Bei der Addition zweier Zahlen im Ein-Byte-Format weiß man normalerweise nicht im voraus, wie das Ergebnis aussehen wird. Da die Summe größer als $FF sein kann, ist es ratsam, für das Ergebnis zwei Bytes bereitzustellen. Sehen wir uns die Beispiele nochmals an:

Hex-Zahlen		Übertrags-Flag	Binär-Zahlen
A7	=		10100111
+ 3E	=		+ 00111110
00E5	=	0	11100101
		Kein Übertrag	
FF	=		11111111
+ FF	=		+ 11111111
01FF	=	1	111111110
		Übertrag	

Das Additionsergebnis ist in beiden Fällen als Zwei-Byte-Zahl dargestellt. Im ersten Fall wurde das Übertragsflag auf 0 zurückgesetzt, da vom achten Bit der Summe aus kein Übertrag durchgeführt wurde. (Das Zwei-Byte-Ergebnis ist $00E5, wobei das höherwertige Byte $00 ist.) Im zweiten Fall ergab sich beim achten Bit jedoch ein Übertrag, und das Übertragsflag wurde gesetzt. Das höherwertige Byte des Ergebnisses beträgt daher $01.

Um das richtige Ergebnis einer Addition zu erhalten, müssen wir den Inhalt des Akkumulators im niederwertigen Byte des Zwei-Byte-Speichers ablegen und das Übertragsflag als das höherwertige Byte des Ergebnisses speichern. Es gibt zwar keine direkte Instruktion für die Speicherung des Übertragsflags, jedoch berücksichtigt der Befehlsaufbau von

ADC diesen Vorgang automatisch. ADC bedeutet: „Addiere den Operanden des Befehls zum Inhalt des Akkumulators plus dem Wert des Übertragsbits, stelle das Ergebnis in den Akkumulator und den Übertrag in das Übertragsregister. Eine Addition läuft daher immer in zwei Stufen ab. In der ersten Stufe wird der aktuelle Status des Übertragsflags verarbeitet und in der zweiten Stufe wird das Flag neu gesetzt.

Da der Stand des Übertragsflags direkten Einfluß auf das Additionsergebnis hat, müssen wir schon vor Ablauf des Befehls den aktuellen Status des Übertragsflags berücksichtigen. Aus diesem Grund haben wir in der vorigen Folge die beiden Befehle CLC und AND A eingesetzt. CLC ist eine Instruktion des 6502 und bedeutet „Übertragsflag löschen", während AND A auf dem Z80 eine logische AND-Verknüpfung des Akkumulators auslöst.

Nachdem die Berechnung abgeschlossen ist, muß der Inhalt des Flags gespeichert werden. Dies läßt sich durch die Addition der Konstanten $00 auf das höherwertige Byte des Ergebnisses erreichen. Ist das Übertragsflag nicht gesetzt, hat dies keine Auswirkung. Ist es gesetzt, erscheint als Übertrag die 1.

Hier der schematische Ablauf unserer ersten Methode der Ein-Byte-Mathematik:

1) Übertragsflag löschen.
2) Akkumulator mit einer Zahl laden.
3) Die zweite Zahl addieren.
4) Den Inhalt des Akkumulators im niederwertigen Byte (LOBYTE) der Zwei-Byte-Adresse speichern.
5) Den Akkumulator mit dem Inhalt des höherwertigen Bytes (HIBYTE) laden.
6) Die Konstante $00 addieren.
7) Den Inhalt des Akkumulators im höherwertigen Byte speichern.

In der Assemblersprache sieht die Routine folgendermaßen aus:

Beide Prozessoren		
Bezeichnung	**Befehl**	**Operand**
BYTE1	EQU	$FF
BYTE2	EQU	$FF
LOBYTE	EQU	$A000
HIBYTE	EQU	$A001
	ORG	$A020

Z80		6502	
Op-code	**Operand**	**Op-code**	**Operand**
LD	A,$00	LDA	#$00
LD	(HIBYTE),A	STA	HIBYTE
AND	A	CLC	
LD	A,BYTE1	LDA	#BYTE1
ADC	A,BYTE2	ADC	#BYTE2
LD	(LOBYTE),A	STA	LOBYTE
LD	A,(HIBYTE)	LDA	HIBYTE
ADC	A,$00	ADC	#$00
LD	(HIBYTE),A	STA	HIBYTE
RET		RTS	

LOBYTE, HIBYTE und ORG enthalten nur Beispielwerte. Die ersten beiden Befehle laden $00 in HIBYTE, um eventuell in dieser Adresse enthaltene Daten zu löschen. LOBYTE braucht nicht auf diese Weise behandelt zu werden, da dessen Inhalt automatisch von dem niederwertigen Byte des Ergebnisses überschrieben wird.

Übertragsflag und andere

Auch in diesem Beispiel kommen die unterschiedlichen Strukturen des Z80 und des 6502 zum Vorschein. Die mnemotischen Kürzel und die Kennzeichnung von Konstanten durch „#" lassen die Bedeutung eines Befehls leicht erkennen. Die Version des Z80 ist nicht so einfach zu durchschauen, da das Kürzel LD für alle Lade- und Speichervorgänge des Akkumulators eingesetzt wird. Auch wird eine Konstante nicht durch „#", sondern durch das Fehlen von Klammern um den Operanden angezeigt. LDA,BYTE1 bedeutet „Lade den Akkumulator mit der Konstanten BYTE1", während LDA,(HIBYTE) „Lade den Akkumulator mit dem Inhalt der Speicherstelle HIBYTE" ausführt. Da in der Assemblersprache alle Befehle im Hexcode eindeutig definiert sind, mag die Frage entstehen, wie der Assembler (oder Programmierer) entscheiden kann, welcher Opcode den gleichlautenden mnemotischen Kürzeln zugeordnet werden muß. Die Antwort dafür liegt in der Art der Adressierung, auf die wir in der nächsten Folge eingehen werden.

Außer dem Übertragsflag enthält das Statusregister des Prozessors noch andere Flags. Hier ein Überblick:

Z80 PSR:	**S**	**Z**		**H**		**P/V**	**N**	**C**
Bitnummer	7	6	5	4	3	2	1	0
MSB								LSB
6502 PSR:	**S**	**V**		**B**	**D**	**I**	**Z**	**C**

PSR BIT	Z80
7	(S)= Vorzeichen
6	(Z)= Null
5	ungenutzt
4	(H)= Halbübertrag
3	ungenutzt
2	(P/V)= Parität/Überlauf
1	(N)= Subtraktion
0	(C)= Übertrag

6502	PSR BIT
(S)= Vorzeichen	7
(V)= Überlauf	6
ungenutzt	5
(B)= Break	4
(D)= BCD-Modus	3
(I)= Interrupt	2
(Z)= Null	1
(C)= Übertrag	0

Im Augenblick sind für uns nur die Flags Übertrag, Vorzeichen und Null wichtig. Wir haben gesehen, daß das Übertragsflag nach einer Addition den Wert des Übertrags aus dem achten Bit des Akkumulators enthält. Das Vorzeichenflag ist immer eine Kopie des achten Bits (Bit 7 des Akkumulators). Das Nullflag wird auf 1 gesetzt, wenn der Inhalt des Akkumulators Null ist und auf 0 zurückgesetzt, wenn der Inhalt nicht Null ist.

Lösung der Assembleraufgaben

1) Die fertig assemblierten Programme werden im nebenstehenden Kasten wiedergegeben. Die Symbole BYTE1 und BYTE2 werden für Konstante und als symbolische Adressen eingesetzt. Als Adressen müssen sie bei der Assemblierung allerdings im Zwei-Byte-Format stehen.

2) Der Befehl „Rückkehren aus einem Unterprogramm" fehlt am Ende beider Programme. Der vollständige Code des 6502 muß folgende Zeile enthalten:

A00D 60 RTS

und die Z80 Version:

A00D C9 RET

3) Der Wert $45 wird zunächst als Konstante in den Akkumulator geladen, dann wird $45 dazu addiert, so daß der Akkumulator den Wert $8A enthält. Das Ergebnis wird in der RAM-Adresse $0045 gespeichert. Der Wert $38 wird dann als Konstante zum Akkumulator addiert, so daß dort der Wert $C2 ($45 + $ 45 + $ 38) enthalten ist. Schließlich wird dieses Gesamtergebnis in der RAM-Adresse $0038 untergebracht.

4) „Konstante" sind Daten, die in einem Befehl selbst gespeichert sind. In den Instruktionen der Beispielprogramme (z. B. LDA #$9C und LD A,$E4) sind die Werte $9C und $E4 Daten, die in den Akkumulator geladen werden. Sie sind als Operanden in den Befehlen gespeichert und stellen den Inhalt der Bytes dar, die dem Op-code jeweils unmittelbar folgen. Daten, die nicht als Konstante in den Op-code integriert werden können, werden in anderen Bereichen des Speichers untergebracht.

5) Der Wert von BYTE1 ist $45. In der korrekten Schreibweise zeigt es die Speicheradresse $0045 an. Die Adresse befindet sich also auf der Speicherseite 0.

Übung

Wenn wir den Inhalt des Prozessor Statusregisters (PSR) untersuchen wollen, ist es ratsam, diese Zahl im Binärformat und nicht als Hexbyte anzuzeigen. Die folgenden Programmzeilen für den Spectrum sind eine Unterroutine zur Umwandlung von dezimalen in binäre Zahlen. Setzen Sie diese Zeilen in das Monitorprogramm ein.

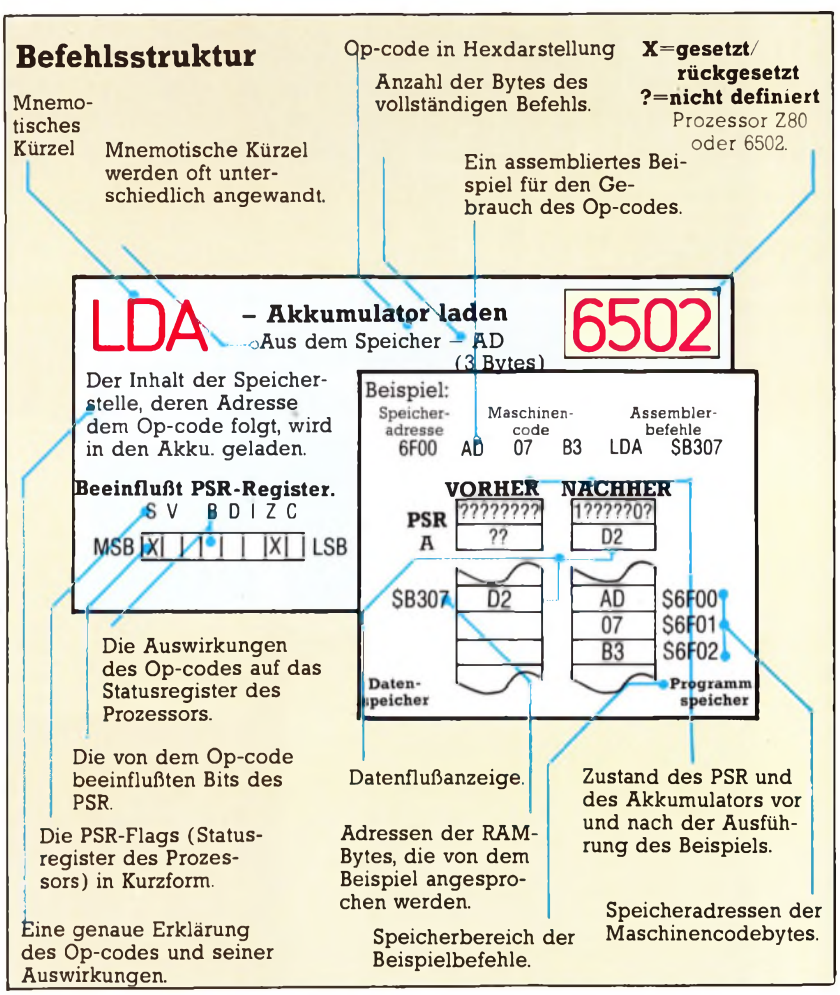
```
7000 RGM ********** BINAER BYTE U/R *******
7001 REM * UMWANDLUNG DER ZAHL N (<256)    *
7002 REM * IN EINE 8—ZEICHEN—BINAER—       *
7003 REM * DARSTELLUNG IN B$
7010 B$=""
7020 FOR D=8 TO 1 STEP-1
7030 LET N1=INT(N/2)
7040 LET R=N-2*N1
7050 LET B$=STR$(R)+B$
7060 LET N=N1
7070 NEXT D
7080 RETURN
```

BASIC-Dialekte

Auf dem Commodore 64 muß die Zeile 7050 folgendermaßen geändert werden:

```
7050 B$=MID$(STR$(R),2)+B$
```

Speicheradresse	Maschinencode	Assemblerbefehle		
6502				
0000		START	EQU	$A000
0000		BYTE1	EQU	$45
0000		BYTE2	EQU	$38
0000			ORG	START
A000	A9 45		LDA	#BYTE1
A002	18		CLC	
A003	69 45		ADC	#BYTE1
A005	8D 45 00		STA	BYTE1
A008	69 38		ADC	#BYTE2
A00A	8D 38 00		STA	BYTE2
Z80				
0000		START	EQU	$A000
0000		BYTE1	EQU	$45
0000		BYTE2	EQU	$38
0000			ORG	START
A000	3E 45		LD	A,BYTE1
A002	A7		AND	A
A003	CE 45		ADC	A,BYTE1
A005	32 45 00		LD	(BYTE1),A
A008	CE 38		ADC	A,BYTE2
A00A	32 38 00		LD	(BYTE2),A



LDA – Akkumulator laden
Konstante – A9 (2 Bytes) — 6502

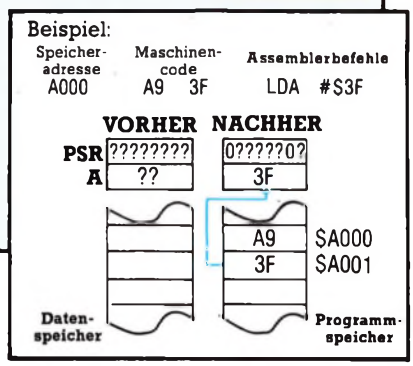
Der Inhalt des Bytes, das dem Op-code folgt, wird in den Akku. geladen.

Bewirkt beim PSR:

S V B D I Z C
MSB [X][ ][ ][ ][ ][X][ ] LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
A000	A9 3F	LDA #$3F



LD A, – Akkumulator laden
Konstante – 3E (2Bytes) — Z80

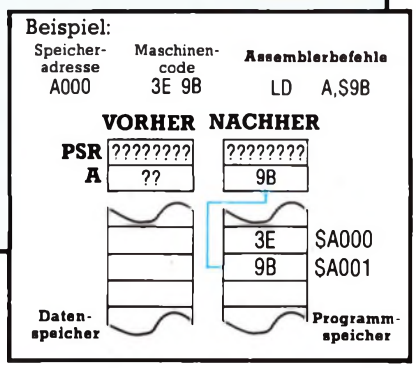
Der Inhalt des Bytes, das dem Op-code folgt, wird in den Akku. geladen.

Bewirkt beim PSR:

S Z H V N C
MSB [ ][ ][ ][ ][ ][ ][ ] LSB
Keine Wirkung

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
A000	3E 9B	LD A,$9B



LDA – Akkumulator laden
vom Speicher – AD (3 Bytes) — 6502

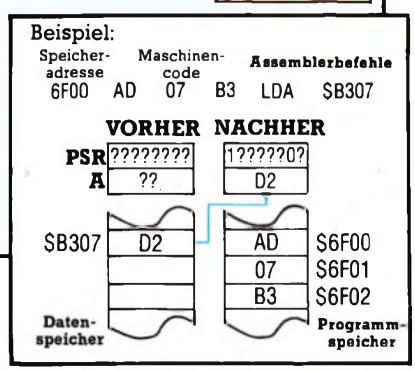
Der Inhalt der Speicherstelle, deren Adresse dem Op-code folgt, wird in den Akku. geladen.

Bewirkt beim PSR:

S V B D I Z C
MSB [X][ ][ ][ ][ ][X][ ] LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
6F00 AD 07 B3	LDA $B307	



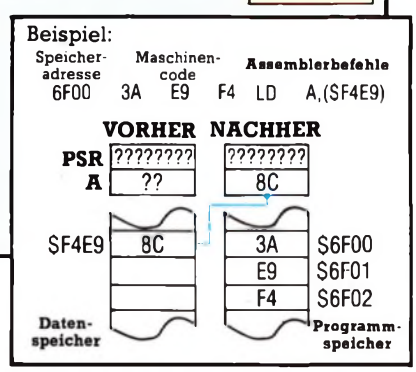
LD A, – Akkumulator laden
vom Speicher – 3A (3 Bytes) — Z80

Der Inhalt der Speicherstelle, deren Adresse dem Op-code folgt, wird in den Akkumulator geladen.

S Z H V N C
MSB [ ][ ][ ][ ][ ][ ][ ] LSB
Keine Wirkung

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
6F00	3A E9 F4	LD A,($F4E9)



STA – Akkumulator speichern
in den Speicher – 8 D (3 Bytes) — 6502

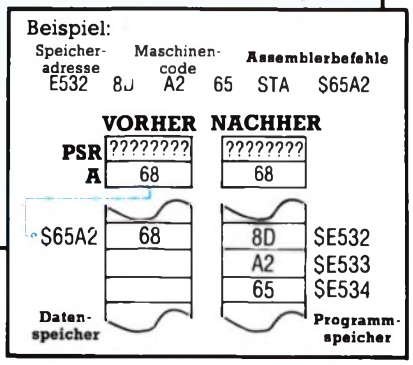
Der Inhalt des Akku. wird in die Speicherstelle geladen, deren Adresse dem Op-code folgt.

Bewirkt beim PSR

S V B D I Z C
MSB [ ][ ][ ][ ][ ][ ][ ] LSB
Keine Wirkung

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
E532 8D A2 65	STA $65A2	



LD( ),A – Akkumulator laden
in den Speicher – 32 (3 Bytes) — Z80

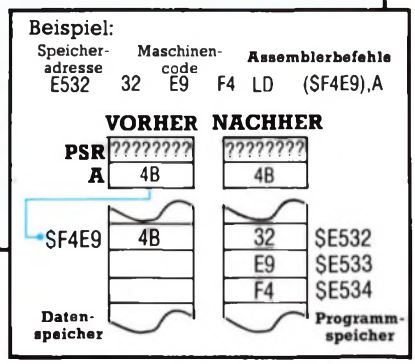
Der Inhalt des Akku. wird in die Speicherstelle geladen, deren Adresse dem Op-code folgt.

Bewirkt beim PSR:

S Z H V N C
MSB [ ][ ][ ][ ][ ][ ][ ] LSB
Keine Wirkung

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
E532	32 E9 F4	LD ($F4E9),A



ADC – Addieren mit Übertrag
Konstante – 69 (2 Bytes) — 6502

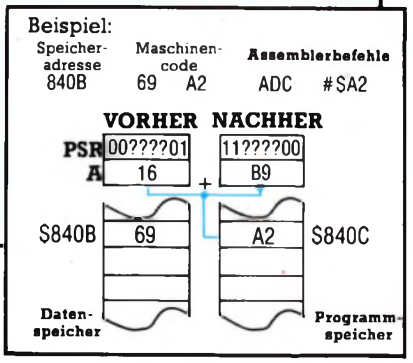
Das Übertragsflag und der Inhalt des Bytes, das dem Op-code folgt, werden zu dem Inhalt des Akku. addiert.

Bewirkt beim PSR

S V B D I Z C
MSB [X][X][ ][ ][ ][X][X] LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
840B	69 A2	ADC #$A2



ADC A, – Addieren mit Übertrag
Konstante – 69 (2 Bytes) — Z80

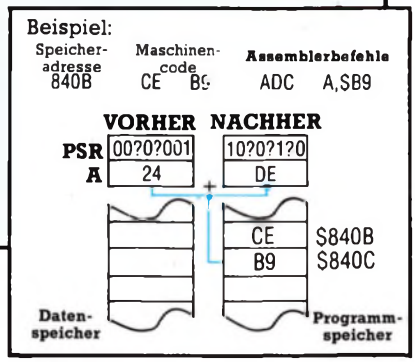
Das Übertragsflag und der Inhalt des Bytes, das dem Op-code folgt, werden zu dem Inhalt des Akku. addiert.

Bewirkt beim PSR:

S Z H V N C
MSB [X][X][ ][X][ ][X] LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
840B	CE B9	ADC A,$B9



Arten der Adressierung

Ausgeklügelte Adressiermethoden unterstützen die Flexibilität der Assemblersprache. In dieser Folge sehen wir uns die verschiedenen Arten der direkten, indirekten und indizierten Adressierung an.

Jeder Assemblerbefehl hat einen direkten oder indirekten Bezug zum Speicherinhalt. Da sich Speicherbytes aber nur durch ihre Adressen voneinander unterscheiden, enthält jeder Befehl mindestens eine Adresse. Die Methode der Adressierung kann dabei direkt und eindeutig sein wie bei dem Befehl LDA $E349 („Lade den Akkumulator mit dem Inhalt der Adresse $E349"). Dabei werden der Akkumulator (ein Speicherbyte, dessen Adresse aus einem Namen statt einer Zahl besteht) und die Adresse $E349 direkt angegeben und ihre Beziehung zueinander durch die Struktur des Befehles definiert.

Der Bezug zu einer Adresse kann aber auch weniger deutlich sein. So scheint der Befehl RET („Rücksprung vom Unterprogramm") keine Adresse zu enthalten. Versteht man den Befehl jedoch als „Die Adresse des nächsten Befehls, der ausgeführt werden soll, befindet sich an der Stelle, von dem aus die letzte Unterroutine aufgerufen wurde", wird deutlich, daß eine Adressierung stattfindet. Die Adresse, deren Inhalt verändert werden soll (d. h. der Programmzähler – das Register, in dem sich die Adresse des nächsten Befehls befindet), wurde allerdings weder direkt angegeben, noch die Adresse, an der sich der neue Inhalt befindet (d. h. die nächste Befehlsadresse).

In dieser Serie haben wir bis jetzt nur zwei Adressiermethoden eingesetzt: Die „unmittelbare" Adressierung, wie in LD A, $45 und ADC #$31, und die „direkte" oder absolute Adressierung, wie in STA $58A7 und LD($696C),A. Mit diesen beiden Methoden scheinen sich (außer der „implizierten" Adressierung wie RTS oder RET) alle Adreßanforderungen erfüllen zu lassen.

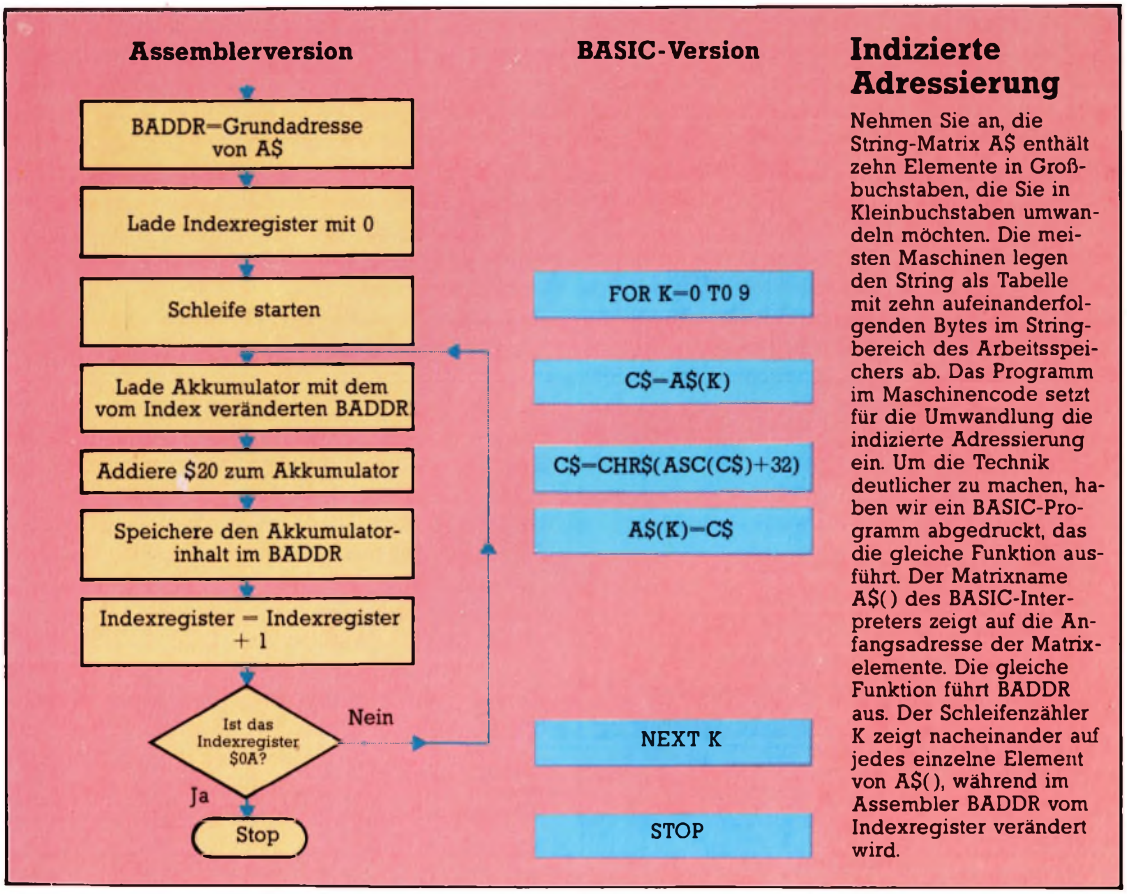
Die Seite 0 des Speichers – auch „Zero Page" genannt – hat für die Adressierung eine besondere Bedeutung. Das Befehlsformat bezieht sich dabei auf Adressen im Bereich $0000 bis $00FF, deren höherwertiges Byte die Zahl $00 ist. Befehle dieser Art bestehen aus nur zwei Bytes: einem für den Op-code und einem für das niederwertige Byte der Adresse. Wenn die CPU eine Ein-Byte-Adresse erhält, setzt sie automatisch $00 als höherwertiges Byte ein und spricht damit Seite 0 an. Der Vorteil der Zero-Page-Adressierung ist ihre hohe Ausführgeschwindigkeit.

Die Zero Page wird von den Microprozessoren Z80 und 6502 unterschiedlich eingesetzt. In der Assemblersprache des 6502 kann jeder Befehl, der den RAM-Bereich adressiert (zum Beispiel LDA), auch im Zero-Page-Format verwandt werden, wobei der CPU alle 256 Bytes dieser Speicherseite zur Verfügung stehen. Im Z80-Assembler gibt es jedoch nur den Befehl RST („Restart" oder „Reset"), der die Seite 0 (und dort auch nur acht Adressen) anspricht. Da RST in seiner Adressierung begrenzt ist und sein Op-code nur aus einem einzigen Byte besteht (die Adresse ist Teil des Bytes), hat er sehr kurze Ausführzeiten.

Die im Bild gezeigte Anzeigetafel eines Bahnhofs enthält Daten, die ein Reisender als Information benötigt. Ein Wegweiser, wo er die Tafel finden kann, ist daher eine indirekte Adressierung dieser Daten. In der Assemblersprache bedeutet indirekte Adressierung, daß die im Operanden des Befehls enthaltene Adresse die Adresse des Bytes enthält, in dem die zu verarbeitenden Daten gespeichert sind. Die Operandenadresse ist ein Zeiger.

Zeit sparen!

Es erscheint kleinlich, die Ausführzeiten verschiedener Assemblerbefehle gegeneinander abzuwägen, da selbst der Ablauf des langsamsten Befehls nur in Mikrosekunden meßbar ist. Es gibt jedoch Assemblerprogramme, bei denen die Einsparung einer Mikrosekunde pro Befehl den Erfolg oder Mißerfolg bestimmt. Spielprogramme mit bewegten, dreidimensionalen Bildern in hoher grafischer Auflösung können zum Beispiel für jedes neue Bild Millionen von Befehlen enthalten, die mit hoher Geschwindigkeit ablaufen müssen, damit die Zügigkeit des Spiels nicht behindert wird. Ein



Indizierte Adressierung

Nehmen Sie an, die String-Matrix A$ enthält zehn Elemente in Großbuchstaben, die Sie in Kleinbuchstaben umwandeln möchten. Die meisten Maschinen legen den String als Tabelle mit zehn aufeinanderfolgenden Bytes im Stringbereich des Arbeitsspeichers ab. Das Programm im Maschinencode setzt für die Umwandlung die indizierte Adressierung ein. Um die Technik deutlicher zu machen, haben wir ein BASIC-Programm abgedruckt, das die gleiche Funktion ausführt. Der Matrixname A$() des BASIC-Interpreters zeigt auf die Anfangsadresse der Matrixelemente. Die gleiche Funktion führt BADDR aus. Der Schleifenzähler K zeigt nacheinander auf jedes einzelne Element von A$(), während im Assembler BADDR vom Indexregister verändert wird.

nennenswerter Zeitgewinn entsteht bereits dann, wenn bei einem Befehl, der in einer Schleife 64 000mal eingesetzt wird, eine Mikrosekunde gespart werden kann.

Die indizierte Adressierung

Die „indizierte" Adressierung ist für die Assemblersprache wichtig, da sie den Aufbau matrixähnlicher Datenstrukturen ermöglicht. Ohne diese Strukturen können Speicherstellen nur einzeln und – wie in unseren Beispielen – mit Angabe der genauen Adresse angesprochen werden.

Die wesentlichen Elemente der indizierten Adressierung sind die Basisadresse und der Index. Wenn eine Datentabelle – zum Beispiel die ASCII-Codes – in fortlaufenden Bytes gespeichert werden soll, dann ist die Adresse des ersten Tabellenbytes die Basisadresse. Die Position jedes Bytes der Tabelle wird nun zu der Basisadresse in Beziehung gesetzt. Das erste Byte steht dabei auf Position 0, das zweite Byte auf Position 1, das dritte auf Position zwei etc. Der Bezug einer Byteposition zu seiner Basisadresse wird Index genannt, wobei sich die absolute Adresse jedes Tabellenbytes aus der Basisadresse und dem Byteindex errechnet.

Auch bei der indizierten Adressierung sind die Assembler des Z80 und des 6502 unterschiedlich strukturiert. Der Chip des 6502 enthält zwei Ein-Byte-Register (X und Y), die je einen Index enthalten, mit dem eine Basisadresse verändert werden kann. Die Länge der Tabelle ist damit auf maximal 256 Bytes begrenzt. Der Z80-Chip dagegen enthält zwei Zwei-Byte-Register (IX und IY), die die Basisadresse selbst enthalten. Die Register können erweitert und reduziert werden und zeigen damit auf die entsprechenden Bytes der Tabelle. Da IX und IY Zwei-Byte-Register sind, läßt sich jedes Byte, das von der CPU angesprochen werden kann, adressieren.

Die indirekte Adressierung

Die „indirekte" Adressierung setzt Pointeradressen ein. Stellen Sie sich als Beispiel einen Filmclub vor, dessen Mitglieder sich jede Woche treffen, um einen Film anzusehen, den der Clubpräsident ausgewählt hat. Da der Film in einem von zwölf Kinos gezeigt werden kann, schreibt der Präsident den Vorführort und die Zeit auf eine Karte, die er in einem Schaufenster des Stadtzentrums aushängt. Wenn die Clubmitglieder erfahren wollen, wo der Film der nächsten Woche gezeigt wird, brauchen sie nur den Laden mit der Information aufzusuchen, der auf den jeweiligen Aufführort „zeigt". Die Adresse des Ladens ist daher indirekt die Adresse des Kinos.

Die Befehle der indirekten Adressierung enthalten somit eine Zeigeradresse, die nicht verarbeitet wird, sondern auf den Inhalt einer anderen Speicherstelle zeigt. Besonders in Verbindung mit der indizierten Adressierung

hat die indirekte Adressierung entscheidende Vorteile. Nehmen Sie an, Sie wollen eine Assemblerroutine schreiben, die eine Datentabelle nach einem bestimmten Zeichen durchsuchen und Ihnen die Indexposition eines bestimmten Zeichens liefern soll. Nehmen Sie weiterhin an, daß in unterschiedlichen Bereichen des Speichers mehrere dieser Tabellen untergebracht sind, die alle mit der gleichen Routine behandelt werden sollen. Wenn Sie in die Routine eine indirekte Adressierung einbauen, bei der der Zeiger auf die Basisadresse der gewünschten Tabelle zeigt, können Sie jede Tabelle ansprechen. Sie müssen vor Aufruf der Routine nur den Zeiger auf die entsprechende Basisadresse setzen.

In der praktischen Anwendung werden statt der Grundbefehle Mischformen eingesetzt:

Befehl	Operand	Einsatzform
LDA	#$34	unmittelbar
LDA	$A2	Zero Page direkt
LDA	$967F	Absolut, direkt
LDA	$A2,X	Zero Page mit X indiziert
LDA	$967F,X	Absolut, mit X indiziert
LDA	$967F,Y	Absolut, mit Y indiziert
LDA	($A2,X)	Indirekt, mit X vorindiziert
LDA	($A2),Y	Indirekt, mit Y nachindiziert

Beachten Sie, daß die beiden letzten Befehlsversionen sowohl die Zero-Page-Adressierung als auch die indirekte und die indizierte Methode einsetzen. Wenn die Tabelle auf den ersten Blick auch etwas verwirren mag, so wird ihre Bedeutung mit ein wenig Übung schon bald deutlich werden, denn auch bei den Befehlen LDA und ADC konnten wir die unmittelbare und absolute Adressierung ohne Probleme einsetzen.

Unterscheidung gleicher Mnemotiks

In der Tabelle ist auch die Antwort auf die Frage enthalten: Wie kann der Op-code eines Befehls bestimmt werden, wenn das mnemotische Kürzel gleich lautet? Die Beispieltabelle zeigt, wie sich die Operandenformate jeder Adressierungsart voneinander unterscheiden. Die einzige Unklarheit entsteht bei der Verwendung symbolischer Adressen. Dabei ist nicht gleich deutlich, ob LDA SYMB1 eine Adresse der Zero Page oder eine absolute Adresse ist. Normalerweise löst das Assemblerprogramm diese Frage automatisch. Bei einer manuellen Assemblierung müssen Sie jedoch überprüfen, ob SYMB1 als Ein- oder Zwei-Byte-Zahl gedacht ist.

Lösung der Aufgaben

Das Monitorprogramm ist für Erweiterungen gedacht, und das Hinzufügen einer weiteren Menüoption ist daher recht einfach:

Die Version für den Spectrum

1) Folgende Änderungen sind bei der Initialisierung zu beachten:

```
1050 LET LT=5:DIM C$(LT):DIM0$
     (LT,24):DIM X$(16)
1150 LET C$="ADGQB":LET C1=-48
     :LET C2=10-CODE(C$(1))
1280 LET 0$(5)=" B-BINAERE ANZEIGE"
```

2) Die Eingaberoutine hat in Zeile 2000 die Anfangsadresse bereits angesprochen, diese in A$ als Hexzahl mit vier Ziffern umgewandelt und in DN als Dezimalzahl gespeichert. Die Unterroutine für die binäre Anzeige sieht folgendermaßen aus:

```
7000 REM**HEX&BIN DISP S/R**
7020 FOR P=DN TO (DN+15)
7040 LET NM=P:GOSUB 3100: PRINT H$,
7060 LET N=PEEK(P):LET NM=N
7080 GOSUB 3000: PRINT B$;" ";
7100 GOSUB 7300: PRINT B$
7120 IF P=65535 THEN LET P=DN+15
7140 NEXT P
7200 RETURN
7300 REM**BINARY BYTE S/R**
7310 LET B$=""
7320 FOR D=8 TO 1 STEP-1
7330 LET N1=INT(N/2)
7340 LET R=N-2*N1
7350 LET B$=STR$(R)+B$
7360 LET N=N1
7370 NEXT D
7380 RETURN
```

Version für den Acorn B/Commodore

Übernehmen Sie die Spectrum-Version mit folgenden Änderungen:

1) Verändern Sie die Initialisierung von LT und 0$ wie bei dem Spectrum und ergänzen Sie die Zeile

```
1150 durch C$(5)="B".
```

2) Zeile 600 übergibt die Programmsteuerung an die entsprechende Unterroutine. Verändern Sie diese auf dem Commodore 64 und auf dem Acorn B folgendermaßen:

```
600 ON CM GOSUB 5000, 5500, 6000,
    6500, 7000
```

3) Ändern Sie auf dem Acorn B die Zeile 7060 in

```
7060 N=?(P):NM=N
```

4) Ändern Sie auf dem Commodore die Zeile 7350 in

```
7350 B$=MID$(STR$(R),2)+B$
```

Assemblerschleifen

Schleifen und bedingte Verzweigungen werden in der Assemblersprache über die relativen Sprungbefehle und die Flags des Prozessor-Status-Registers gesteuert. Mit diesen Strukturen und der indizierten Adressierung lassen sich auch Datentabellen erstellen und bearbeiten.

Bevor wir uns den verschiedenen Adressierarten der CPU (und den indizierten Adressen) zuwenden, müssen Sie wissen, wie Schleifen funktionieren. In der Assemblersprache gibt es keine automatischen Strukturen wie beispielsweise FOR. . .NEXT in BASIC (ein Befehl des Z80 ist dieser Struktur jedoch sehr ähnlich). Der Ablauf läßt sich jedoch aus Befehlen zusammensetzen, die Entscheidungen fällen oder Bedingungen enthalten und so den Ablauf des Programms beeinflussen können.

Entscheidungsprozesse der Assemblersprache drehen sich hauptsächlich um die Flags des Prozessor-Status-Registers. Diese Bedingungsflags zeigen an, welche Auswirkungen der letzte Befehl auf den Akkumulator hatte. Der Zustand des Nullflags (Z) und des Übertragsflags (C) bestimmt, ob der nächste Befehl ausgeführt oder zu einer anderen Stelle des Programms verzweigt wird. Dabei „entscheidet" der Prozessor, ob er die in seinem Programmzähler gespeicherte Adresse annimmt oder verändert. Der Programmzähler enthält immer die Adresse des Maschinencodes, der als nächstes ausgeführt werden soll, wobei der Prozessor den Op-code aus dem Byte lädt, auf das der Programmzähler zeigt. Die im Programmzähler enthaltene Adresse wird nach dem Laden der Instruktion um die Anzahl der Befehlsbytes erhöht, so daß sie immer auf den Op-code des nächsten Befehls zeigt. Wenn diese Adresse durch einen Befehl verändert wird, entsteht ein Programmsprung.

Verzweigungsbefehle

Der 6502-Befehl BEQ veranlaßt eine Veränderung des Programmzählers, wenn das Nullflag gesetzt ist. BCS ist die entsprechende Instruktion für das Übertragsflag (auf dem Z80 JR Z und JR C). Diese vier Op-codes werden Verzweigungsbefehle genannt. Ihr Operand – eine Ein-Byte-Zahl – wird auf die Adresse des Programmzählers addiert und erzeugt so eine neue Adresse. Sehen wir uns diesen Vorgang genauer an:

ORG $5E00			
6502		Z80	
5E00	ADC #$34	ADC	A,$34
5E02	BEQ $03	JR	Z,$03
5E04	STA $5E20	LD	($5E20),A
5E07	RTS	RET	

Wenn der in $5E00 gespeicherte ADC-Befehl im Akkumulator das Ergebnis Null erzeugt, dann verursachen die Befehle BEQ und JR Z in $5E02, daß $03 zu dem Inhalt des Programmzählers addiert wird. Der nächste auszuführende Befehl ist dann der Rücksprung zu $5E07, und die Instruktion in $5E04 wird dabei übersprungen.

Symbolische Adressen

Auf den ersten Blick scheint dies ein Fehler zu sein, da der Befehl in $5E02 $03 auf den Programmzähler addiert und die dort gespeicherte Adresse daher $5E05 sein müßte. Hier ist es wichtig zu berücksichtigen, daß der Programmzähler immer auf den **nächsten** Befehl zeigt und nicht auf den, der augenblicklich ausgeführt wird. Wenn also die Instruktion in $5E02 mit der Ausführung beginnt, dann enthält der Programmzähler schon die Adresse $5E04 – die Speicherstelle des darauffolgenden Befehls. So ergibt die Addition von $03 auf $5E04 das Ergebnis $5E07, also die Adresse des nächsten Befehls.

Interessant ist hierbei auch, daß der Prozessor nicht überprüfen kann, ob die Adressen korrekt sind. Wenn wir aus Versehen die zu addierende Zahl mit $02 angegeben haben, erhöht sich der Programmzähler um $02 (vorausgesetzt, der Akkumulator enthält Null), und der Prozessor nimmt $5E06 als die Adresse des nächsten Op-Codes. In unserem Programm enthält $5E06 den Wert $5E, das höherwertige Byte des Operanden der Instruktion in $5E04. Der Prozessor kann jedoch nicht beurteilen, ob dieser Befehl richtig ist oder nicht. Für ihn ist $5E ein gültiger Op-Code, wobei er die darauffolgenden Bytes als Operanden ansieht. Das Programm wird vermutlich abbrechen. Die falsche Berechnung von Abständen ist einer der häufigsten Fehler der Maschinencodeprogrammierung.

In der Assemblersprache ist die Abstandsberechnung jedoch kein Problem, da der Assembler diese Aufgabe übernehmen kann. Statt eine Hexadezimalzahl als Operanden des Verzweigungsbefehls anzugeben, können wir eine symbolische Adresse als Zieladresse des Sprungs angeben. Der Assembler wandelt die symbolische Adresse in eine absolute um, berechnet den Abstand zu dieser Adresse und

schreibt ihn in den Maschinencode. Die symbolische Adresse wird „Label" genannt; sie entspricht etwa der Zeilennummer eines BASIC-Programms.

Labels sind alphanumerische Strings, die am Anfang des Assemblerprogramms stehen. Das Assemblerprogramm interpretiert sie als Zwei-Byte-Symbole, die die Adresse des ersten Befehlsbytes repräsentieren. Wir können daher das Programm folgendermaßen umschreiben:

ORG $5E00		
	6502	Z80
5E00	ADC #$34	ADC A,$34
5E02	BEQ EXIT	JR Z,EXIT
5E04	STA $5E20	LD ($5E20),A
5E07 EXIT	RTS	RET

Der Befehl in $5E02 kann jetzt folgendermaßen verstanden werden: „Wenn (IF) der Wert des Akkumulators Null ist, dann springe (THEN GOTO) auf die Adresse, die von dem Label EXIT dargestellt wird".

Mit Labels und Verzweigungsbefehlen können wir jetzt eine Schleife aufbauen:

ORG $5E00		
	6502	Z80
5E00 START	ADC #$34	ADC A,$34
5E02	BNE START	JR NZ,START
5E04	STA $5E20	LD ($5E20),A
5E07 EXIT	RTS	RET

Hier treten das Label START und ein neuer Verzweigungsbefehl auf: BNE bedeutet „Verzweige, falls der Akkumulator ungleich Null ist"; JR NZ bedeutet „Springe, wenn der Akkumulator nicht Null ist". Dieser Code bedeutet: Auf den Akkumulator wird $34 addiert. Falls das Ergebnis nicht Null ist, verzweigt das Programm auf $5E00 – die Adresse des Labels START. $34 wird nochmals auf den Akkumulator addiert, wobei das Ergebnis wiederum entscheidet, ob eine weitere Verzweigung ausgeführt werden soll. Diese „Schleife" setzt sich so lange fort, bis die Verzweigungsbedingung erfüllt ist. Wenn der Inhalt des Akkumulators nach dem ADC-Befehl Null ist, dann findet in $5E02 keine Verzweigung statt, und der Befehl in $5E04 wird ausgeführt.

Es stellt sich die Frage, ob der Akkumulator jemals Null sein kann, da er bei jedem Schleifendurchlauf um weitere $34 erhöht wird. Die Antwort ist einfach: Der Akkumulator ist ein Ein-Byte-Register. Bei einem Zwei-Byte-Ergebnis wird das Übertragsflag des Prozessor-Status-Registers gesetzt, wobei der Akkumulator das niederwertige Byte der Summe enthält. Wenn im Akkumulator beispielsweise $CC gespeichert ist und darauf $34 addiert wird, ist das Ergebnis $0100. Das Übertragsflag wird gesetzt, und der Akkumulator enthält $00 als niederwertiges Byte des Ergebnisses. Da jetzt der Inhalt des Akkumulators Null ist, wird auch das Nullflag gesetzt.

Statt des Nullflags kann auch der Status des Übertragsflags abgefragt werden:

ORG $5E00		
	6502	Z80
5E00 START	ADC #$34	ADC A,$34
5E02	BCC START	JR NC,START
5E04	STA $5E20	LD ($5E20),A
5E07 EXIT	RTS	RET

In dieser Version bedeutet der Befehl in $5E02 „Verzweige nach START, wenn das Übertragsflag nicht gesetzt ist". Sobald das Additionsergebnis größer als $FF ist, wird das Übertragsflag gesetzt, und die Verzweigung nach START wird nicht durchgeführt.

Im Augenblick fehlt jedoch noch ein Schleifenzähler, durch den man beispielsweise die Information erhält, wie oft eine Schleife ausgeführt wurde, bevor die Endbedingung erfüllt war. Dieses läßt sich leicht erreichen, wenn das Indexregister der CPU den Zähler enthält und ein Befehl den Zähler nach jeder Schleife aktualisiert:

6502		
0000	ORG	$5DFD
5DFD	LDX	#$00
5DFF START	INX	
5E00	ADC	#$34
5E02	BCC	START
5E04	STX	$5E20
5E07 EXIT	RTS	

Z80		
0000	ORG	$5DFA
5DFA	LD	IX,$0000
5DFE START	INC	IX
5E00	ADC	A,$34
5E02	JR	NC,START
5E04	LD	($5E20),IX
5E08 EXIT	RET	

Schleifenzähler

Die neue Struktur hat in dem Programm mehrere Veränderungen bewirkt. Zunächst enthalten die Befehle am Anfang des Programms eine neue ORG-Adresse. Diese Instruktionen haben auf dem 6502 und dem Z80 zwar die gleichen Auswirkungen, die Längen und Adressen sind jedoch verschieden. Weiterhin haben neue Versionen der Lade- (LDX, LD IX) und Speicherbefehle (STX, LD ( ), IX) den Anfangswert $00 in das Indexregister der CPU geladen. Das X-Register des 6502 ist ein Ein-Byte-Register, das IX-Register des Z80 hat jedoch ein Zwei-Byte-Format. Die Indexregister haben zwar Spezialfunktionen, sind aber wie der Akkumulator im wesentlichen RAM-Speicher. Wir verwenden sie hier als zusätzliche Akkumulatoren, in denen wir den Schleifenzähler unterbringen. Nach Ende der Schleife

wird der Inhalt des X-Registers (6502) in $5E20 gespeichert, während in der Z80-Version das niederwertige Byte des (Zwei-Byte-)IX-Registers in $5E20 untergebracht wird und das höherwertige Byte in $5E21.

Auch an die Stelle von ADC am Anfang (START) der Schleife sind neue Instruktionen getreten: INX und INC IX sind „Inkrementierbefehle", die den Inhalt des Indexregisters um $02 erhöhen (inkrementieren). Sie aktualisieren den Wert des Schleifenzählers bei jedem Durchlauf.

Berechnungsbeispiele

Das Programm lautet jetzt folgendermaßen: "Stelle den Schleifenzähler auf Null, beginne die Schleife mit der Inkrementierung des Zählers und addiere $34 auf den Akkumulator. Verzweige an den Anfang der Schleife, wenn das Übertragsflag Null ist, andernfalls speichere den Schleifenzähler in $5E20".

6502

```
0000          ORG   $5DFA
5DFA          LDX   #$00
5DFC START    STA   $5E22,X
5DFF          INX
5E00          ADC   #$34
5E02          BCC   START
5E04          STX   $5E20
5E07 EXIT     RTS
```

Z80

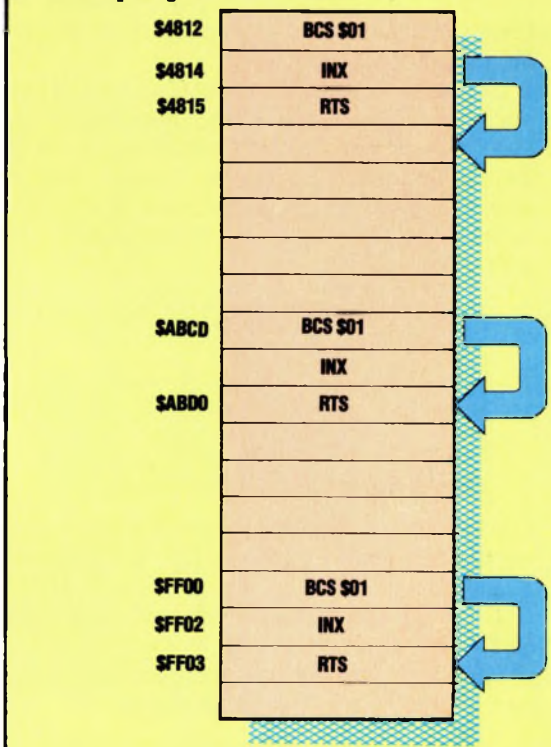
```
0000          ORG   $5DF7
5DF7          LD    IX,$5E00
5DFB START    LD    (IX+$22),A
5DFE          INC   IX
5E00          ADC   A,$34
5E02          JR    NC,START
5E04          LD    ($5E20),IX
5E08 EXIT     RET
```

Hier haben die Versionen des 6502 und des Z80 die gleiche Wirkung: Sie legen an der Adresse $5E22 eine Tabelle der Akkumulatorwerte an, die bei der Ausführung des Programms entstehen, und speichern den Endwert des Schleifenzählers in $5E20. Dieser Endwert entspricht auch der Anzahl der Tabellenbytes (beginnend bei $5E22).

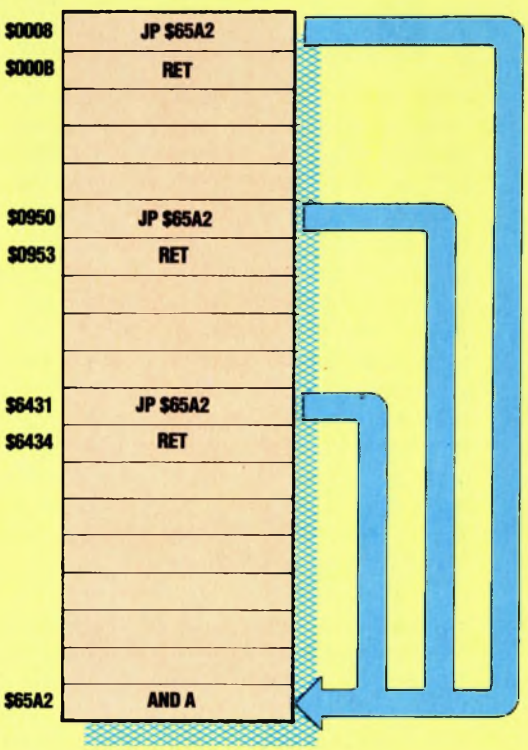
Die Version des 6502 setzt für diese Aufgabe den Befehl STA $5E22,X ein. Er bedeutet: „Addiere den Inhalt des X-Registers auf die Basisadresse $5E22, und speichere dann den Inhalt des Akkumulators auf die neue Adresse". Der STA-Befehl ist hier absolut und direkt adressiert. Das heißt, das X-Register wird als Index eingesetzt, um die Basisadresse $5E22 zu verändern. Da das X-Register anfänglich mit $00 geladen und bei jeder Schleife erhöht wird, ist der Anfangswert des Akkumulators in $5E22 gespeichert, der nächste Wert in $5E23 etc. Nach Beendigung der Schleife legt STX den Endwert des Schleifenzählers in $5E20 ab.

Die Z80-Version verwendet das IX-Register

Relative Sprünge



Absolute Sprünge



Relative Sprünge
Die meisten Verzweigungsbefehle wie BCS („Verzweige, wenn das Übertragungsflag gesetzt ist") und JR NZ („Verzweige, wenn der Akkumulator nicht Null ist") arbeiten mit dem Inhalt des Prozessor-Status-Registers, wenn durch relative Sprünge der Fluß des Programms umdirigiert werden soll. Die Alternative ist der absolute Sprung. In dem nebenstehenden Beispiel bewirkt der Befehl BCS $01 ständig einen relativen Vorwärtssprung von einem Byte (falls die entsprechende Bedingung erfüllt ist) – unabhängig von der Adresse, an der der Maschinencode gespeichert ist. Dem Befehl BCS $01 folgt der Ein-Byte-Befehl INX. Ist das Übertragsflag gesetzt, dann verursacht BCS das Überspringen von INX.

Absoluter Sprung
In diesem Beispiel löst der Befehl JP $65A2 in jedem Fall einen absoluten Sprung aus. Die Programmsteuerung wird dabei auf die Adresse seines Operanden geleitet – hier $65A2. Es wird kein Test ausgeführt, und auch die Adresse, von dem aus der Sprung ausgeführt wurde, ist nicht entscheidend. Die Programmausführung setzt sich immer von der angegebenen Adresse an fort.

Beide Sprungarten haben Vor- und Nachteile. Das wichtigste Kriterium bei der Wahl zwischen einem relativen und einem absoluten Sprung ist die Plazierung: Assemblerroutinen werden oft mit einer bestimmten ORG-Adresse assembliert und später mit einer anderen ORG-Adresse nochmals verwandt. Wenn die Routine nur relative Sprünge enthält, funktioniert sie ohne Änderung der Speicheradressen. Absolute Sprünge können bei einer Änderung der ORG-Adresse die Steuerung jedoch zu Adressen senden, die für die Routine keine Bedeutung haben. Relative Sprünge lassen sich versetzen, absolute nicht.

als Zeiger, der die aktuelle Speicheradresse angibt, während das niederwertige Byte von IX als Schleifenzähler dient. Der Befehl LD IX, $5E00 lädt die Basisadresse $5E00 in das IX-Register, so daß das niederwertige Byte von IX $00 enthält. Die seltsam aussehende Instruktion LD (IX+$22), A bedeutet „Addiere die in IX enthaltene Adresse auf $22, und speichere den Inhalt des Akkumulators auf die dadurch entstehende Adresse". Da IX mit $5E00 initialisiert ist und in der Folge bei jeder Schleife erhöht wird, ist damit der Anfangswert des Akkumulators in $5E22 gespeichert, der nächste Wert in $5E23 usw. Das niederwertige Byte von IX enthält dabei die Anzahl der Durchgänge, die nach Ende der Schleife in $5E20 untergebracht wird. LD (IX+$22),A ist hier eine absolute indirekte und indizierte Adressierung, die zwar komplizierter aber flexibler als der Befehl des 6502 ist.

Übungen

In dieser Folge sind wir auf viele wichtige Punkte eingegangen, die Sie zunächst vielleicht irritieren werden. Die neuen Adressierarten und Befehlsstrukturen lassen sich nur verstehen, wenn sie angewandt werden.

Assemblieren und speichern Sie die verschiedenen Programmteile dieser Folge mit dem „CHAMP"-Assemblerpaket. Untersuchen Sie die bei der Ausführung eines Programms angesprochenen Speicherbytes mit der <DEBUG>-Schaltung. Dabei lohnt es sich, diese Bytes vor dem Programmaufruf mit einer leicht erkennbaren Konstanten – zum Beispiel $FF – zu initialisieren, damit Sie nach dem Ablauf erkennen können, wie sich der Speicherinhalt verändert hat. Nehmen Sie dafür entweder den „<DEBUG> Alter"- oder den „<DEBUG> Move"-Befehl.

Bitte beachten Sie, daß die in den Programmen angegebenen Adressen nur Beispiele sind. Die Programme funktionieren nur, wenn Sie die für Ihre Maschine geeigneten Adressen verwenden.

Laden und Speichern von CHAMP
Bitte legen Sie auf einer anderen Cassette zunächst eine Sicherheitskopie von CHAMP an. Bei den folgenden Instruktionen bezieht sich der LOAD-Befehl auf die CHAMP-Cassette und SAVE auf die Kopie:

Acorn B
1) LOAD"CHAMP"
2) SAVE"CHAMP":RUN: Auf BASIC schalten
3) *SAVE"CHAMP M/C" 1000, 4600

Commodore 64
1) LOAD"CHAMP"
2) SAVE"CHAMP":RUN: eingeben <DEBUG> Modus
3) [w] [ret] drücken, gefolgt von s für SAVE
4) Anfangsadresse 1000; Endadresse 4600; Dateiname "CHAMP M/C"

Spectrum
1) LOAD"CHAMP"
2) Auf BASIC schalten: SAVE"CHAMP" LINE 1
3) SAVE"CHAMP M/C"CODE 27000, 9231

Sie haben gesehen, daß die bedingte Verzweigung vom Inhalt des Prozessor-Status-Registers abhängt. Mit der Binäranzeige des Monitors können Sie den Inhalt des PSR vor und nach der Ausführung eines Befehls ansehen und beobachten, wie sich die Flags verändern. Da es in der Assemblersprache des 6502 und des Z80 keinen Befehl gibt, der den Inhalt des PSR speichert, müssen wir folgende Codes einsetzen:

Z80	
3E00 F5	PUSH AF
3E01 F5	PUSH AF
3E02 E1	POP HL
3E03 22 lo hi	LD (STORE1),HL
3E06 F1	POP AF
6502	
3E00 48	PHA
3E01 08	PHP
3E02 48	PHA
3E03 08	PHP
3E04 68	PLA
3E05 8D lo hi	STA STORE1
3E08 68	PLA
3E09 8D lo' hi'	STA (1+STORE1)
3E0C 28	PLP
3E0D 68	PLA

Diese Befehlsfolge verursacht, daß der augenblickliche Inhalt des PSR in einem Byte gespeichert wird, das mit STORE1 addressiert wird, während der Inhalt des Akkumulators in (1+STORE1) untergebracht ist. Sie brauchen diese Befehle nur als Block vor oder hinter den Programmteil zu setzen, den Sie beobachten wollen. Beachten Sie aber, daß Sie für jeden Einsatz dieses Blocks die Zahl Zwei auf den Wert von STORE1 addieren müssen. Bei der Ausführung eines Programms können Sie sich nun mit dem Monitor den Teil des Speichers ansehen, in dem die Inhalte des PSR und des Akkumulators abgelegt wurden.

Es liegt nahe, diesen Block als Unterroutine zu behandeln, statt ihn mehrfach zu speichern. In der Assemblersprache gibt es einen Befehl, der dem GOSUB in BASIC entspricht. Sein Gebrauch würde den Ablauf jedoch komplizierter machen, da er den Stack anspricht, der auch von einigen Befehlen des Programmblocks eingesetzt wird (PLA, PUSH und PHP verändern den Stack, auf den wir später genauer eingehen werden). Für die unterschiedlichen Codelängen des Z80 und 6502 sind die Zwei-Byte-Register des Z80 und die zugehörigen Befehle verantwortlich.

Universell einsetzbar

Maschinencodemodule lassen sich für die unterschiedlichsten Programme einsetzen, wenn sie statt absoluter Adressen und fester Werte Symbole und Lables enthalten und dadurch „frei verschiebbar" (relocatable) werden. In dieser Folge stellen wir weitere Assemblerbefehle vor und werfen einen ersten Blick auf die Methoden, mit denen die Assemblersprache Unterroutinen aufruft.

Viele Grundfunktionen, die im Befehlssatz einer Hochsprache als selbstverständlich angesehen werden, sind im Assembler nicht von vornherein vorhanden. Da die Assemblersprache hauptsächlich „primitive" Anweisungen enthält, die die „einfach denkende" CPU verstehen kann, müssen Routinen für grundlegende Aufgaben, wie Ein- und Ausgabesteuerung oder Zwei-Byte-Arithmetik, bei jeder Programmierung neu angelegt werden. Es bietet sich an, die am häufigsten eingesetzten Routinen auf Papier, Band oder Diskette zu sammeln, um sie bei Bedarf in neue Programme einbinden zu können.

Hier gibt es jedoch zwei grundlegende Probleme. Zunächst müssen wichtige Routinen so aufgebaut sein, daß sie ohne Änderung in den unterschiedlichsten Programmen funktionieren können. Diese Module dürfen auch nicht an bestimmte Speicheradressen gebunden sein, sondern müssen bei einer Neuassemblierung unter einer anderen ORG-Adresse genau die gleichen Aufgaben erfüllen wie an ihrer ursprünglichen Speicheradresse.

Beide Probleme sind auch dem BASIC-Programmierer vertraut, der seine Programme auf mehreren Maschinentypen einsetzen will. Die Lösung der Assemblersprache sieht der BASIC-Lösung ähnlich: Variablen übertragen die Werte des Hauptprogramms auf die Unterroutinen, die durch die Verwendung eigener (lokaler) Variablen unabhängig sind. Außerdem wird auf die Angabe von festen Werten (numerische und String-Konstanten) und auf Zeilennummern verzichtet.

Im bisherigen Verlauf des Kurses haben wir die Inhalte der Speicherstellen ähnlich wie BASIC-Variablen behandelt. Aus Gründen der Einfachheit wurden die Bytes dabei fast ausschließlich mit absoluten Adressen eingesetzt. Soll eine Routine jedoch universell einsetzbar sein, müssen statt absoluter Adressen und fester Werte Symbole verwandt werden, die die Pseudo-Op-codes des Assemblers als Variablen und als Zeilennummern des Programms anbieten. Hier ein Beispiel für beide Anwendungen:

6502			Z80		
DATA1	EQU	$12	DATA1	EQU	$12
DATA2	EQU	$79	DATA2	EQU	$79
	LDA	DATA1	LD	A,(DATA1)	
LOOP	ADC	DATA2	ADC	A,(DATA2)	
	BNE	LOOP	JR	NZ,LOOP	
	RTS		RET		

Die Tabelle enthält zwei Symboltypen: zwei Werte und ein Label. Alle Symbole sind Operanden von Assemblerbefehlen. Das Programm kann somit leicht modifiziert werden. Die einzigen absoluten Werte sind DATA1 und DATA2.

Außer den bereits erwähnten Pseudo-Opcodes gibt es noch DB, DW und DS (die Bezeichnungen können wie bei ORG und EQU von Assembler zu Assembler verschieden sein). Diese drei Anweisungen bedeuten „Definiere Byte", „Definiere Wort" und „Definiere Speicher".

			ORG	$D3A0
D3A0	5F	LABL1	DB	$5F
D3A1	CE98	LABL2	DW	$98CE
D3B3		LABL3	DS	$10
D3B3		DATA1	EQU	LABL3

Symboltabelle:
LABL1=D3A0 : LABL2 = D3A1 : LABL3 = D3A3
DATA1 = D3A3
ASSEMBLIERUNG BEENDET — KEINE FEHLER

In diesem vollständigen Assemblerlisting (die Anzeige eines Assemblerprogramms) taucht zum ersten Mal eine Symboltabelle auf, die die im Programm eingesetzten Symbole und ihre entsprechenden Werte enthält. Die neuen Abläufe haben folgende Aufgaben: Die Zeile, die mit LABL1 anfängt, enthält den Pseudo-Op DB. Aus dem Programmlisting läßt sich ersehen, daß LABL1 durch die ORG-Anweisung die Adresse $D3A0 zugeordnet wird. DB setzt den Wert $5F auf das von LABL1 adressierte Byte. Die Op-codespalte zeigt, daß die Speicherstelle $D3A0 mit $5F initialisiert wurde.

LABL2 stellt die Adresse $D3A1 dar. DW initialisiert jedoch ein „Wort" (zwei aufeinander folgende Bytes) und bringt damit den Wert $98CE in den Speicheradressen $D3A1 und $D3A2 im lo-hi-Format unter. Da DW seine Operanden automatisch in das lo-hi-Format umwandelt, wird er oft für die Initialisierung

von Zeigeradressen eingesetzt. LABL2 oder die Speicherstelle $D3A1 könnte eine derartige Adresse sein, die auf $98CE zeigt.

Der Befehl DS $10 addiert $10 auf den Programmzähler. Dies ist in der Symboltabelle leichter zu erkennen als im Programmlisting: LABL3 stellt die Adresse $D3A3 (die Adresse, die dem vorherigen Befehl folgt) dar, obwohl dieser Wert aufgrund der Liste $D3B3 zu sein scheint. $D3B3 ist jedoch die Speicheradresse des nächsten Befehls. Damit wird deutlich, daß DS $10 einen Block von 16 Bytes (von $D3A3 bis einschließlich $D3B2) zwischen sich und dem nächsten Befehl reserviert.

Bei dem letzten Befehl des Listings setzt EQU ein Symbol dem Wert eines anderen gleich. DATA1 entspricht daher $D3A3 (dem Wert von LABL3). Auch dieser Ablauf kann zunächst etwas verwirren. LABL3 ist die symbolische Darstellung der Adresse $D3A3. DATA1 EQU LABL3 bedeutet also „das Symbol DATA1 soll die gleiche Bedeutung und den gleichen Wert wie das Symbol LABL3 erhalten". Die Tatsache, daß DB den Inhalt von $D3A3 auf $5F gesetzt hat, hat keinen Einfluß auf die Bedeutung der Symbole LABL3 und DATA1.

Pseudo-Befehle

Auf den ersten Blick scheint DB der Anweisung EQU zu entsprechen. Dies ist jedoch nicht der Fall. LABL1 bedeutet: „die Speicheradresse $D3A0", während DB $5F dieses Byte mit dem Wert $5F initialisiert. Obwohl der Wert von LABL1 festgelegt ist, kann der Inhalt der Adresse, die LABL1 darstellt, jederzeit geändert werden (zum Beispiel durch Speichern des Akkumulatorinhalts später im Programm). DATA1 dagegen ist ein Symbol, dessen Wert von der Anweisung EQU festgelegt wurde und durch das Programm nicht mehr zu ändern ist. LABL3 zeigt nun wiederum auf den Anfang eines Datenbereiches mit 16 Bytes, deren Inhalt das Programm ändern kann. Die Adresse von LABL3 ist jedoch fixiert.

Damit sind die Möglichkeiten der neuen Pseudo-Befehle aber noch nicht erschöpft. Sehen Sie sich folgende neue Version des vorigen Programmteils an:

```
                        ORG   $D3A0
D3A0   4D4553   LABL1   DB    'MESSAGE1'
D3A9   CE98     LABL2   DW    $98CE
D3BB            LABL3   DS    $10
D3BB            DATA1   EQU   LABL3

Symboltabelle:
LABL1 = D3A0: LABL2 = D3A9: LABL3 = D3AB
DATA1 = D3AB
ASSEMBLIERUNG BEENDET — KEINE FEHLER
```

Operand der DB-Anweisung ist der String „MESSAGE1". Der Assembler hat die Speicherstellen von $D3A0 bis $D3A8 mit den ASCII-Werten der von Anführungsstrichen eingeschlossenen Zeichen initialisiert. Die Spalte der Speicheradressen macht dies deutlich: Der Inhalt der drei Bytes von $D3A0 bis $D3A2 wird mit $4D, $45 und $53 angegeben – den Hexzahlen der ASCII-Codes für die Buchstaben „M", „E" und „S".

Diese Einrichtung befreit den Programmierer nicht nur von der mühsamen Übersetzung von Zeichen und Meldungen in Folgen von ASCII-Codes, sondern macht auch das Programm leichter lesbar. Das Modul deutet außerdem an, daß Assemblerprogramme auch Bildschirmanzeigen enthalten können und Ergebnisse nicht immer im Speicher abgelegt werden müssen, damit sie dort mit dem Monitorprogramm betrachtet werden können. Im weiteren Verlauf dieser Serie werden wir auch die Anzeigemöglichkeiten der Assemblersprache behandeln. Bevor wir uns aber diesem Bereich zuwenden, muß auf einige andere Abläufe eingegangen werden. Da Sie jedoch wissen, daß aus Speichertabellen aufgebaute Bildschirmanzeigen nichts anderes sind als bestimmte Speicherbereiche, können Sie auch schon den Bildschirm adressieren.

Das Wichtigste der neuen DB-Anweisung ist jedoch, daß LABL1 damit der Status einer BASIC-String-Variablen zugeordnet wird. Die BASIC-Anweisung

```
200 LET A$="MESSAGE1"
```

richtet eigentlich einen Zeiger auf den Anfang einer Bytetabelle ein, die die ASCII-Codes für

Übungen

1) Im ersten Programmteil des Haupttextes reserviert der Pseudo-Op-code DS $10 Speicherbytes von der Adresse an, die durch LABL1 dargestellt wird. Schreiben Sie ein Assemblerprogramm, das die Zahlen $0F bis $00 in diesem Block in absteigender Reihenfolge speichert – eine Zahl pro Byte. Eine Schleife mit indizierter Adressierung sollte die Aufgabe ausführen. Zu diesem Zweck benötigen Sie den Befehl DEX (Register X verringern) oder DEC (IX+0) (IX erweitern). Die Schleife sollte ausgeführt werden, solange das Indexregister nicht das Setzen des Nullflag auslöst. Verwenden Sie daher die Verzweigungsbefehle BNE oder JR NZ.

2) Schreiben sie nach der Methode der ersten Übung ein Programm, das eine Meldung kopiert, die von dem Pseudo-Op-code unter LABL1 abgelegt wurde (siehe zweiter Programmteil des Haupttextes). Die Kopie soll in einen Speicherblock gelegt werden, dessen Anfangsadresse von dem Pseudo-Op-code in LABL2 gespeichert wurde. Da die Adresse $98CE sich für Ihren Computer nicht eignen mag, muß hier eventuell die Initialisierung geändert werden. Das Programm sollte für alle Adressen und für Meldungen jeder Länge funktionieren. Verwenden Sie daher die Zeichenzahl als Schleifenzähler, oder bauen Sie in das Programm eine Endabfrage der Meldung ein – zum Beispiel durch einen Stern als letztes Zeichen jeder Meldung.

„M", „E", „S" enthält. Findet der BASIC-Interpreter einen Bezug auf A$, dann sucht er in seiner Symboltabelle nach dem Zeiger der entsprechenden Speicheradresse, das heißt nach der Anfangsadresse des Inhalts von A$. In unserem Assemblerprogramm können wir LABL1 wie A$ behandeln, vorausgesetzt, wir haben ein Modul, das mit der Methode der indizierten Adressierung eine Speichertabelle bearbeiten kann.

Maschinencode für GOSUB

Durch Pseudo-Op-codes können wir absolute Adressen und feste Werte durch Symbole ersetzen und damit die meisten Schwierigkeiten für den universellen Einsatz unserer Programme beseitigen. Wir müssen diese frei verschiebbaren Module nun nur noch von unserem Hauptprogramm aus adressieren können, das heißt, wir brauchen einen Maschinencodebefehl, der dem GOSUB des BASIC entspricht.

Die Befehle JSR und CALL führen diesen Sprung für den 6502 und den Z80 aus. Beide Befehle benötigen eine absolute Adresse (die ein Label sein kann) als Operanden, und beide ersetzen den Inhalt des Programmzählers durch die Adresse, die ihr Operand angibt. Der nächste Befehl, der zur Ausführung kommt, ist dann die erste Anweisung der adressierten Unterroutine. Die Programmausführung setzt sich solange fort, bis sie einen Rücksprungbefehl findet (RTS oder RET). Der Rücksprung ersetzt den augenblicklichen Inhalt des Programmzählers durch die Adresse, die vor Ausführung des JSR- oder CALL-Befehls dort vorhanden war. Der nächste Befehl, der ausgeführt wird, ist daher die Anweisung, die JSR oder CALL unmittelbar folgt. Auch der BASIC-Interpreter benutzt diesen Mechanismus bei der Ausführung und der Rückkehr von einem GOSUB-Befehl. Hierbei entsteht jedoch die Frage, wie der ursprüngliche Inhalt des Programmzählers beim Aufruf des Rücksprungs wiederhergestellt werden kann. Die Antwort ist einfach: Die Befehle JSR und CALL schieben (push) den Inhalt des Programmzählers auf den Stapel (stack), bevor sie ihn durch die Adresse der Unterroutine ersetzen. RTS oder RET ziehen diese Adresse vom Stapel wieder herunter (pop) und setzen sie in den Programmzähler ein. Was ein Stapel ist, welche Funktion er ausübt und wie sich Werte dort ablegen oder herunterziehen lassen, werden wir in der nächsten Folge behandeln. Anhand von weiteren Übungen können Sie danach Ihre Kenntnisse überprüfen.

Befehlssatz

BEQ – **Verzweigen bei Null** — Relativ – FO (2 Bytes) — **6502**

Wenn das Nullflag gesetzt ist, wird der Programmzähler um den Wert des Bytes erhöht, das dem Op-code unmittelbar folgt.

Beeinflußt PSR-Register

	S	V		B	D	I	Z	C
MSB								LSB

KEINE AUSWIRKUNG

Speicheradresse	Maschinencode	Beispiel: Assemblerbefehle
8F00	FO 16	BEQ $16

	Vorher		Nachher
Programmzähler	02	lo	18
	8F	hi	8F

Programmspeicher: FO, 16, $8F00

JR Z – **Verzweigen bei Null** — Relativ – 28 (2 Bytes) — **Z80**

Wenn das Nullflag gesetzt ist, wird der Programmzähler um den Wert des Bytes erhöht, das dem Op-code unmittelbar folgt.

Beeinflußt PSR-Register

	S	Z		H		V	N	C
MSB								LSB

KEINE AUSWIRKUNG

Speicheradresse	Maschinencode	Beispiel: Assemblerbefehle
8F00	28 16	JR Z,$16

	Vorher		Nachher
Programmzähler	02	lo	18
	8F	hi	8F

Programmspeicher: 28, 16, $8F00

INX – **Register X erweitern** — implizit – F8 (1 Byte) — **6502**

Der Inhalt des Registers X wird um 1 erhöht.

Beeinflußt PSR-Register

	S	V		B	D	I	Z	C
MSB	X						X	LSB

Speicheradresse	Maschinencode	Beispiel: Assemblerbefehle
F391	E8	INX

	Vorher	Nachher
PSR	????????	0??????1?
X	FF	00

Programmspeicher: E8, $F391

INC IX – **Register IX erweitern** — implizit – DD 23 (2 Bytes) — **Z80**

Der Inhalt des Registers IX wird um 1 erhöht.

Beeinflußt PSR-Register

	S	Z		H		V	N	C
MSB								LSB

KEINE AUSWIRKUNG

Speicheradresse	Maschinencode	Beispiel: Assemblerbefehle
F391	DD23	INC IX

	Vorher		Nachher
Programmzähler	FF	lo	00
	E7	hi	E8

Programmspeicher: DD, 23, $F391

Last In First Out

Der Stack (Stapel) ist ein fest definierter Speicherbereich des Computers. Er steht der CPU für „Notizen" zur Verfügung und spielt beim Ablauf von Unterroutinen eine wichtige Rolle. In dieser Folge erklären wir Aufgaben und Funktionsweise des Stacks.

Die Speicherverwaltung ist die Grundlage der Assemblerprogrammierung. Fast alle bisher untersuchten Befehle übertragen Daten in bestimmte Speicherstellen oder laden sie von dort. Speicherstellen können über verschiedene Adressiermethoden angesprochen werden, wobei der Operand fast immer eine Speicheradresse enthält. Es gibt jedoch eine Befehlsklasse, die ohne Adreßoperanden auf den Speicher zugreift. Diese Befehle arbeiten mit einem Speicherbereich, der als Stack (Stapel) bezeichnet wird.

Der Stack steht der CPU und dem Programmierer als Kurzzeitspeicher zur Verfügung. Er ist eine Art „Notizblock", auf dem Daten notiert werden, die schnell gelesen und auch schnell wieder gelöscht werden können. Die Stackbefehle kopieren Daten der CPU-Register in freie Stapelbereiche oder laden sie wieder in die CPU-Register zurück. Da der Stack Pointer (Stapelzeiger) – selbst ein CPU-Register – immer die Adresse des nächsten freien Stackbytes enthält, benötigen diese Befehle keinen Adreßoperanden. Alles, was auf dem Stack abgelegt werden soll, wird automatisch auf das Byte geschrieben, dessen Adresse der Stack Pointer enthält, während alle Daten, die vom Stack heruntergezogen werden, immer von dem Byte des Stacks kommen, das zuletzt beschrieben wurde.

Der Stack des 6502 besteht aus den 256 RAM-Bytes von $0100 bis $01FF, wogegen im Z80 Systembereich und Größe des Stacks variabel sind und vom Betriebssystem bestimmt werden. Der Stack Pointer des 6502 besteht aus einem Byte, der des Z80 aus zwei Bytes.

Der Stack Pointer

Der 6502 sieht den Inhalt des Stack Pointers als niederwertiges Byte der Stackadresse an, auf das automatisch $01 – das „neunte Bit" des Stack Pointers – als höherwertiges Byte addiert wird. Da das zusätzliche Bit immer gesetzt (bzw. „Eins") ist, liegen alle Stackadressen des 6502 auf der Seite eins des Speichers.

Mit dem Stack Pointer des Z80 lassen sich Adressen zwischen $0000 und $FFFF (der gesamte Adreßraum des Z80) ansprechen. Der Stack kann daher an jeder beliebigen Stelle des RAMs liegen. Eine Verlegung des Stapelbereichs ist jedoch nicht zu empfehlen, da das Betriebssystem diesen Bereich unmittelbar nach dem Laden festlegt und dort alle Daten speichert.

Die folgende Routine tauscht die Inhalte der Speicherstellen LOC1 und LOC2 gegeneinander aus:

6502		Z80	
LDA	LOC1	LD	A,(LOC1)
PHA		PUSH	AF
LDA	LOC2	LD	A,(LOC2)
STA	LOC1	LD	(LOC1),A
PLA		POP	AF
STA	LOC2	LD	(LOC2),A

Der Inhalt von LOC1 wird in den Akkumulator geladen und von dort auf den Stack gelegt (PUSH). Danach wird der Inhalt von LOC2 in den Akkumulator geladen und auf LOC1 kopiert. Durch Laden des obersten Stackbytes in den Akkumulator (POP) befindet sich nun der ursprüngliche Inhalt von LOC1 im Akkumulator. Das Kopieren des Akkumulatorinhalts auf LOC2 beendet das Programm. Bemerkenswert ist dabei, daß der Stackbefehl den Inhalt von LOC1 ohne Angabe einer Speicheradresse (nur durch die implizite Angabe des nächsten freien Stackbytes) speichert.

Dieses Programm gibt Einblick in die Methodik des Stacks. Außer der Umkehrung des Vorgangs beim Laden und Lesen fällt auf, daß alle Daten sequentiell bearbeitet werden: Zuletzt auf den Stack gelegte Daten werden als erste wieder heruntergezogen. Mehrere Informationen, die nacheinander – ohne dazwischenliegende Leseoperation – auf den Stack geschrieben werden (PUSH), sind in aufeinanderfolgenden Adressen des Stacks abgelegt.

Stellen Sie sich den Stack als eine Anzahl Karten vor, die Sie mit Notizen beschreiben und auf Ihrem Schreibtisch stapeln. Im weiteren Verlauf nehmen Sie die jeweils oberste Karte herunter, lesen diese und werfen sie weg. Die Karte, die Sie als letztes beschrieben haben, liegt immer oben auf dem Stapel. Die Datenstruktur des Stapels wird daher auch als LIFO (Last In – First Out: Das zuletzt Eingegebene wird zuerst wieder ausgegeben) bezeichnet. Die Umkehrung dieser Datenstruktur nennt man FIFO (First In – First Out, die zuerst eingegebenen Daten werden auch zuerst wieder ausgegeben).

In unserem Programmbeispiel wurden ebensoviele Daten auf den Stack geschoben wie heruntergezogen werden. Auf dieses Gleichgewicht braucht zwar nicht unbedingt Rücksicht genommen zu werden, doch können Unterroutinen bei Nichtbeachtung dessen leicht falsche Rücksprungadressen erhalten, die zum Programmabbruch führen. Dieser häufig auftretende Fehler läßt sich jedoch vermeiden, wenn man die Zahl der PUSH- und POP-Befehle miteinander vergleicht.

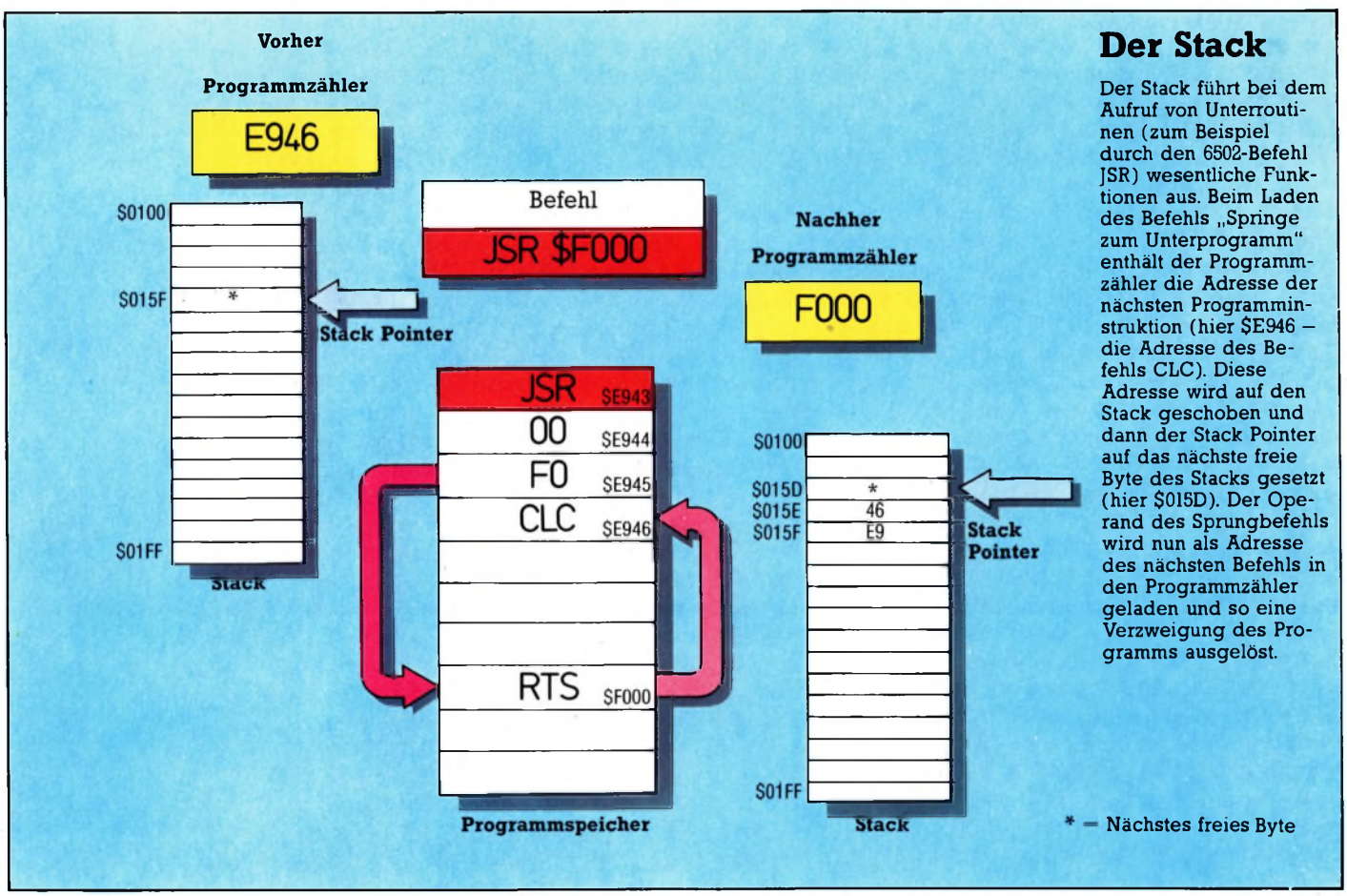
Zwei-Byte-Register des Z80

Die Version des Z80 unterscheidet sich in einem wesentlichen Aspekt von der des 6502: Während der 6502 nur Ein-Byte-Register auf den Stack schieben kann, speichert der Z80 dort ausschließlich Zwei-Byte-Register. Wenn Sie mit PUSH oder POP den Akkumulator des Z80 ansprechen, greifen Sie auch auf das Prozessor-Status-Register zu, da die CPU diese beiden (Ein-Byte-) Register als ein Zwei-Byte-Register mit dem Namen AF (Akkumulator-Flag) behandelt. Die universelle Einsetzbarkeit des Z80 beruht zum großen Teil auf seiner Fähigkeit, Zwei-Byte-Register verarbeiten zu können.

Es ist guter Programmierstil, den Inhalt aller CPU-Register am Anfang einer Unterroutine auf den Stack zu schieben und sie vor dem Rücksprung wieder zurückzuladen. Auf diese Weise stellen Sie sicher, daß die CPU nach dem Rücksprung den gleichen Zustand hat wie vorher, und brauchen außerdem nicht zu befürchten, daß wichtige Daten des Hauptprogramms verändert wurden. Sehen Sie sich folgendes Unterprogramm an:

	6502		Z80	
	LDA	LOC1	LD	A,LOC1
SUM	ADC	#$6C	ADC	A,$6C
GSUB	JSR	SUBR0	CALL	SUBR0
TEST	BNE	SUM	JR	NZ,SUM
EXIT	RTS		RET	
SUBR0	PHP		PUSH	AF
	PHA		PUSH	HL
	TXA		PUSH	DE
	PHA		PUSH	BC
	TYA		PUSH	IX
SUBR1	PHA		PUSH	IY
SUBR2	STA	LOC2	LD	(LOC2),A
	LDA	#$00	LD	A,$00
SUBR3	PLA		POP	IY
	TAY		POP	IX
	PLA		POP	DE
	TAX		POP	BC
	PLA		POP	HL
SUBR4	PLP		POP	AF
	RTS		RET	

Die Befehle zwischen SUBR0 und SUBR1 schieben die aktuellen Registerinhalte auf den



Der Stack

Der Stack führt bei dem Aufruf von Unterroutinen (zum Beispiel durch den 6502-Befehl JSR) wesentliche Funktionen aus. Beim Laden des Befehls „Springe zum Unterprogramm" enthält der Programmzähler die Adresse der nächsten Programminstruktion (hier $E946 – die Adresse des Befehls CLC). Diese Adresse wird auf den Stack geschoben und dann der Stack Pointer auf das nächste freie Byte des Stacks gesetzt (hier $015D). Der Operand des Sprungbefehls wird nun als Adresse des nächsten Befehls in den Programmzähler geladen und so eine Verzweigung des Programms ausgelöst.

Stack, während die Instruktionen zwischen SUBR3 und SUBR4 sie wieder in die entsprechenden Register laden. Die beiden auf SUBR2 folgenden Befehle sind der eigentliche Inhalt der Routine, wobei der zweite Befehl gegenstandslos ist, da die darauf folgenden Instruktionen den Akkumulator wieder verändern.

Beachten Sie hierbei, daß die Z80-Befehle PUSH und POP jedes Registerpaar als Operanden enthalten können, während sich auf dem 6502 nur der Akkumulator (PHA und PLA) und das Prozessor-Status-Register (PHP und PLP) ansprechen lassen. Der 6502 braucht daher zusätzliche Befehle für die Datenübertragung zwischen Registern und Akkumulator (TXA, TAX, TYA, TAY). Beachten Sie weiterhin, daß wir in die Version des Z80 absichtlich einen Fehler eingebaut haben, indem wir nicht alle Register in umgekehrter Reihenfolge vom Stack herunterziehen. Hier wird deutlich, mit welcher Vorsicht Stackvorgänge programmiert werden müssen. Das Beispiel zeigt auch, wie der Stack den Inhalt eines Registers in ein anderes übertragen kann – eine aufwendige, aber zuweilen sehr praktische Methode, um zwischen Registern Daten auszutauschen.

In der nächsten Folge gehen wir auf die Funktionen und Einsatzmöglichkeiten der CPU-Register ein und schließen damit die allgemeine Einführung ab.

Lösung der Übungen unserer vorigen Folge

1) Diese Unterroutine speichert in absteigender Reihenfolge die Zahlen $0F bis $00 in einem Block von $10 Bytes, der von dem Pseudo-Op-code DS bei LABL1 reserviert wurde.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DS $10	LABL1	DS $10
LABL2	DW $7100	LABL2	DW $C100
		OFFST	EQU $0F
BEGIN	LDY #$FF	BEGIN	LD IX,LABL1
	LDX #$10		LD B,OFFST
LOOP0	INY	LOOP0	LD (IX+0),B
	DEX		INC IX
	TXA	ENDLP0	DJNZ LOOP0
	STA LABL1,Y		LD (IX+0),B
ENDLP0	BNE LOOP0		RET
	RTS		

Der 6502 nimmt das Y-Register als Index der Adresse LABL1, während das X-Register als Schleifenzähler und als Ursprung der zu speichernden Daten dient. Beachten Sie, daß das X-Register bereits zwei Befehle vor dem BNE-Test bei ENDLP0 dekrementiert wird. Da die Befehle TXA („Übertrage Register X in den Akkumulator") und STA keinen Einfluß auf das Prozessor-Status-Register haben, verändert sich das Dekrementierungsergebnis dadurch nicht.

Die Z80-Version setzt als Zeiger der Speicheradresse das indirekt adressierte IX ein und nimmt das B-Register als Zähler und Datenquelle. Neben dem ENDLP0 der Z80-Version sehen wir den Befehl DJNZ LOOP0, der „Dekrementiere das Register B und führe einen relativen Sprung aus, falls das Ergebnis nicht Null ist" bedeutet.

2) Diese Routine kopiert die in LABL1 gespeicherte Meldung auf einen Block, der bei der in LABL2 gespeicherten Adresse beginnt. Der Wert $0D (der ASCII-Code für Return oder Enter) ist am Ende der Meldung als Abschlußzeichen gespeichert.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DB 'THIS IS A MESSAGE'	LABL1	DB 'THIS IS A MESSAGE'
TERMN8	DB $0D	TERMN8	DB $0D
LABL2	DW $7100	LABL2	DW $C100
CR	EQU $0D	CR	EQU $0D
ZPLO	EQU $FB		
BEGIN	LDA LABL2	BEGIN	LD IX,LABL1
	STA ZPLO		LD IY,(LABL2)
	LDA LABL2+1	LOOP0	LD A,(IX+0)
	STA ZPLO+1		LD (IY+0),A
	LDY $FF		INC IX
LOOP0	INY		INC IY
	LDA LABL1,Y		CP CR
	STA (ZPLO),Y	ENDLP0	JR NZ,LOOP0
	CMP CR		RET
ENDLP0	BNE LOOP0		
	RTS		

Die Version des 6502 verwendet das Y-Register im nach-indizierten indirekten Modus als Index der indirekten Adresse ZPLO. Diese Adressierungsmethode ist nur mit dem Y-Register möglich und erfordert eine Operandenadresse, die auf der Seite Null liegt – daher die Initialisierung von ZPLO und ZPLO+1 mit der in LABL2 gespeicherten Adresse. Die Betriebssysteme von 6502er Maschinen setzen normalerweise fast alle Speicheradressen der Seite Null ein. Da auf dem Commodore 64 die Adressen von $FB bis $FF und auf dem Acorn B die Adressen von $70 bis $8F jedoch ungenutzt sind, wird ZPLO auf eine dieser Speicherstellen gelegt. Die Z80-Version verwendet IX im indizierten Modus und IY im indizierten indirekten Modus.

Beide Routinen verwenden einen Befehl zum Akkumulatorvergleich – CMP CR (6502) und CP CR (Z80) – bei dem die Flags des Prozessor-Status-Registers (PSR) durch eine Subtraktion des Operanden vom Akkumulatorinhalt beeinflußt werden. Wenn der Akkumulator $0D enthält, wird das Nullflag gesetzt. Dadurch ist bei ENDLP0 die Bedingung für eine Verzweigung nicht erfüllt, und die Steuerung wird an den Rücksprungbefehl übergeben.

Übungen

1) Ändern Sie die zweite Programmroutine in den Lösungen so, daß die Meldung von LABL1 in umgekehrter Reihenfolge wieder in LABL1 gespeichert wird:

LABL1 EGASSEM
A SI SIHT

Verwenden Sie dazu den Stack.

2) Ändern Sie die Routine wiederum, so daß die Worte der Meldung in ihrer ursprünglichen Reihenfolge bleiben, die Zeichen jedes Wortes jedoch umgekehrt werden.

LABL1 SIHT SI
A EGASSEM

Adreßregister

Bisher wurde untersucht, wie die CPU den Speicher über Register wie den Akkumulator und die ALU anspricht. Diesmal gehen wir auf die grundlegenden mathematischen Operationen ein.

Die Unterschiede zwischen den Microprozessoren Z80 und 6502 zeigen sich am deutlichsten in der völlig andersartigen Ausführung einfacher mathematischer Vorgänge. Die vielen Register und der hochentwickelte Satz von Befehlen kennzeichnen den Z80 als komplex und flexibel, während der 6502 durch die Einfachheit seiner Struktur und den unkomplizierten Befehlssatz eher ein simpler Prozessor zu sein scheint, der zwar robust und praktisch ist, aber nicht die Klasse des Z80 besitzt. Obwohl dieser Eindruck teilweise durchaus berechtigt sein kann, besitzt der 6502 mit seinen vielen Adressierungsmodi und seinem Einsatz der Zero-Page als zusätzlichem Index-Register doch eine Flexibilität, mit der er auf dem Markt für Computer noch lange Zeit eine wichtige Rolle spielen wird.

Der Z80 besitzt einen wesentlichen Vorteil in der Anpassungsfähigkeit seiner Register. Sie lassen sich wahlweise als Zwei-Byte- und als Ein-Byte-Register einsetzen und verfügen damit über einen enormen Adreßraum. Doch obwohl der 6502 keine Zwei-Byte-Register besitzt, kann er über bestimmte Adressierungsarten die Zero-Page als eine Matrix von Ein-Byte- und Zwei-Byte-Registern einsetzen.

Wir haben gesehen, daß die CPU-Register viele Möglichkeiten bieten, den Speicher zu adressieren. Für die Bearbeitung des Speichers sind normalerweise jedoch andere Vorgänge wichtig als das Laden, Speichern und Vergleichen einzelner Bytes. So spielt die Ausführung der vier Grundrechenarten in einem Computersystem eine wesentliche Rolle. Da die Befehlssätze des Z80 und des 6502 jedoch nur Addition und Subtraktion unterstützen, müssen Multiplikation und Division sowie die Addition und Subtraktion von Zahlen größer als $FF programmiert werden. Diese Begrenzung beider CPUs und die damit verbundene Notwendigkeit, Algorithmen für Multiplikation und Division erarbeiten zu müssen, kann für den Programmierer großen Lernwert haben.

Mit den Befehlen ADC und INC hatten wir auf beiden CPUs eine Ein-Byte-Arithmetik ausgeführt. Die folgenden Programme addieren den Inhalt zweier Zwei-Byte-Speicherstellen auf zwei unterschiedliche Weisen:

6502			Z80		
ADDR1	DW	$7E60	ADDR1	DW	$7E60
ADDR2	DW	$4A51	ADDR2	DW	$4A51
SUM	DS	$03	SUM	DS	$03
BEGIN	CLC		BEGIN	LD	A,$00
	LDA	ADDR1		AND	A
	ADC	ADDR2		LD	HL,(ADDR1)
	STA	SUM		LD	DE,(ADDR2)
	LDA	ADDR1+1		ADD	HL,DE
	ADC	ADDR2+1		LD	(SUM),HL
	STA	SUM+1		ADC	A,$00
	LDA	$00		LD	(SUM+2),A
	ADC	$00		RET	
	STA	SUM+2			
	RTS				

Während die Ein-Byte-Methode des 6502 auch auf dem Z80 funktioniert, läßt sich die Methode des Z80 nicht auf dem 6502 einsetzen, da Registerpaare verwendet werden. Beachten Sie die Strategien, mit denen ein möglicher Übertrag abgefangen wird. Zwischen CLC (6502) und AND A (Z80), die das Übertragsflag vor der Addition auf 0 setzen, und der Veränderung des dritten Bytes von SUM befindet sich eine ganze Reihe von Vorgängen, die das größtmögliche Ergebnis berücksichtigen.

Der Subtraktionsvorgang ähnelt der Addition, da beide Prozessoren über einen SBC („Subtrahieren mit Übertrag") -Befehl verfügen, wobei die Zwei-Byte-Subtraktion allerdings nur auf dem Z80 unterstützt wird. Da hier ein negatives Ergebnis möglich ist, müssen wir untersuchen, wie ein algebraisches Zeichen binär dargestellt werden kann.

Zunächst brauchen wir über negative Zahlen nur folgende Aussage zu treffen:

Wenn A + B = 0, dann A = -B

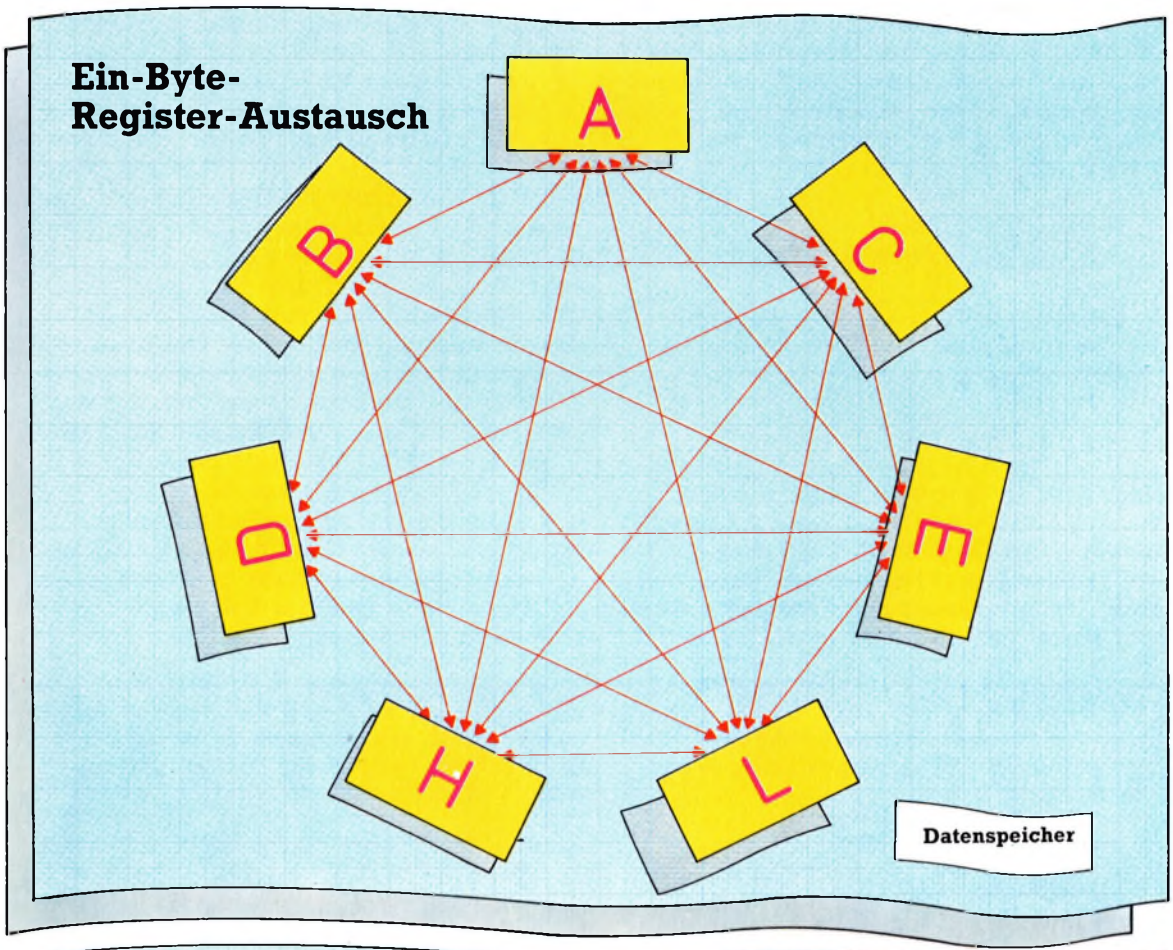
Falls A eine positive Zahl darstellt, dann ist ihre Umkehrung (oder Komplement) die Zahl, die zu A addiert Null ergibt. So ist zum Beispiel das Ein-Byte-Komplement der Ein-Byte-Zahl $04 die Zahl $FC:

$04 + $FC = $100

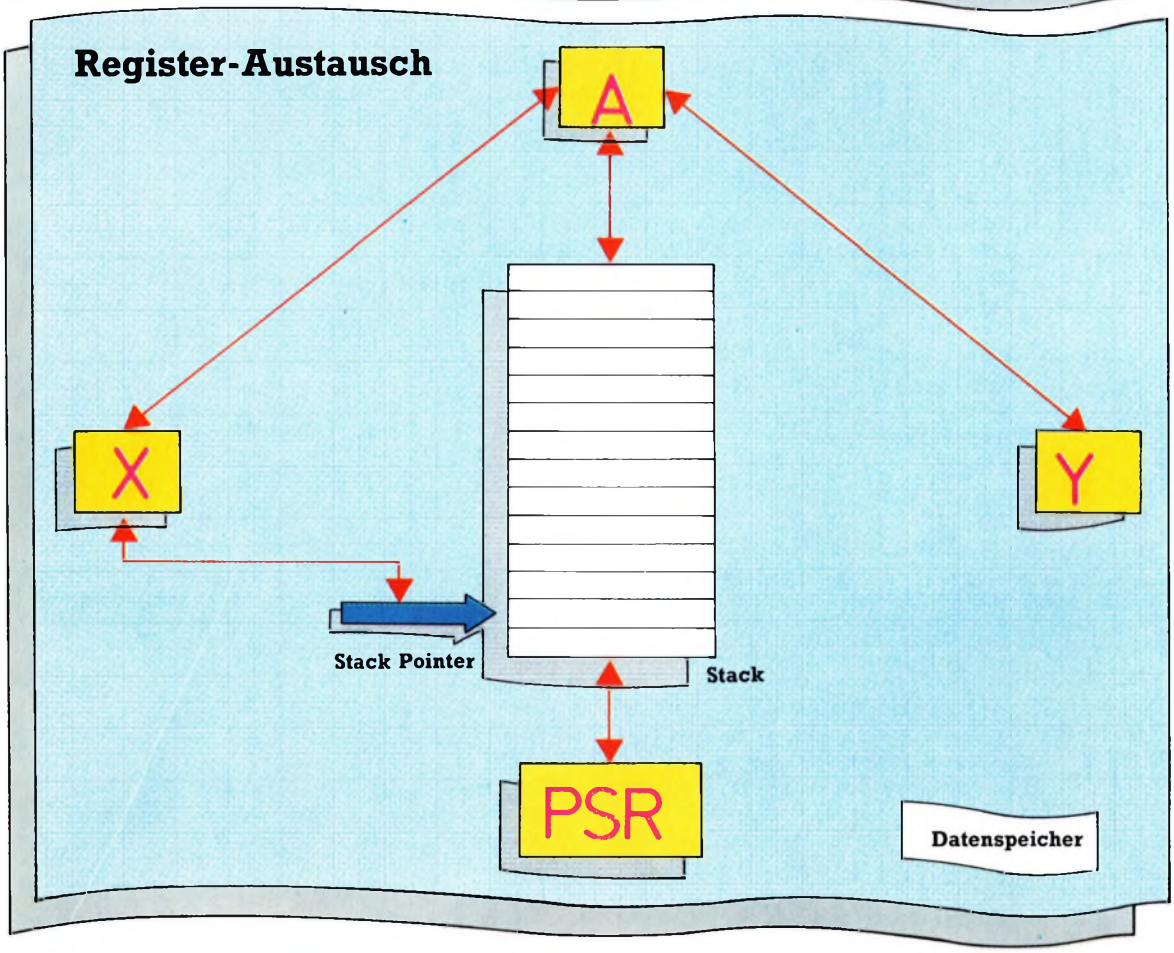
Bedenken Sie hierbei, daß $100 = $00 ergibt, falls nur ein Ein-Byte-Register für das Ergebnis zur Verfügung steht. Durch die Komplementdarstellung läßt sich eine Subtraktion auch als eine Addition mit negativem Vorzeichen ansehen, wie unser Beispiel verdeutlicht:

A − B bedeutet dasselbe wie A + (−B)

$08−$05 ist daher dasselbe wie $08 + (−$05). Da (−$05) sich aber auch als $FB schreiben läßt ($FB + $05 ergibt $100), kann unsere ursprüngliche Subtraktion als $08 + $FB dargestellt werden. Die Summe ist $103 und im Ein-Byte-Format $03.



Die Datenregister des Z80 haben als Ein-Byte-Register die Möglichkeit, mit jedem anderen Ein-Byte-Register Daten auszutauschen. Sie können den Speicher in der direkten, unmittelbaren, indirekten, absoluten und indizierten Adressierung ansprechen. Bei ihrem Einsatz als Zwei-Byte-Registerpaare BC, DE, HL können sie 16-Bit-Daten mit dem Speicher und dem Stack austauschen und auch als 16-Bit-Akkumulatoren für Addition und Subtraktion dienen. Diese vielfältigen Einsatzmöglichkeiten sind der Schlüssel zu dem großen Erfolg des Z80.



Die interne Kommunikation des 6502 ist streng linear und beschränkt sich auf die Datenübertragung im Acht-Bit-Format. Nur der Akkumulator kann direkt mit X und Y Daten austauschen. Nur X kann den Stack-Pointer ansprechen, und nur das PSR und der Akkumulator können auf den Stack zugreifen. Der Speicher läßt sich auf absolute, direkte, indirekte, indizierte, unmittelbare Art und über die Zero-Page ansprechen. Der flexible Einsatz der Zero-Page gleicht die geringe Kapazität des Registersatzes aus. Sie stellt 128 CPU-Register im Zwei-Byte-Format zur Verfügung.

Diese Art der Darstellung wird Zweier-Komplement genannt. Das Zweier-Komplement einer Ein-Byte-Zahl entsteht durch die Subtraktion dieser Zahl von $100. Einer- und Zweier-Komplement stehen in folgender Beziehung zueinander:

```
$05 = 00000101 binär
$FA = 11111010 Einer-Komplement
          +1
-------------------------------
$FB = 11111011 Zweier-Komplement

$05 + $FA = $FF

$05 + $FB = $00
```

Das Einer-Komplement einer Ein-Byte-Zahl entsteht durch Umkehrung jedes binären Bits der Zahl. Wird Eins zu dem Ergebnis addiert, entsteht das Zweier-Komplement. Die Addition einer Zahl und seines Einer-Komplements ergibt immer $FF, während eine Zahl und ihr Zweier-Komplement als Ergebnis immer $00 (eigentlich $100) haben. In der Mathematik der ganzen Zahlen mit Vorzeichen ist es daher üblich, die Zahlen von $00 bis $7F als positiv anzusehen (0 bis 127) und $80 bis $FF als negativ (−128 bis −1). In der Binärdarstellung dieser Zahlen werden Sie bemerken, daß alle negativen Ganzzahlen das Bit 7 gesetzt haben, während bei den positiven Zahlen Bit 7 immer auf 0 steht. Bit 7 wird daher auch als Vorzeichen-Bit bezeichnet, wobei das Übertragsflag des Prozessor-Status-Registers als Kopie von Bit 7 gesetzt oder zurückgesetzt wird und damit über das Ergebnis des letzten arithmetischen oder logischen Vorgangs Auskunft gibt.

Dieses recht komplizierte Thema läßt sich leider nicht umgehen, wenn Sie mit positiven und negativen Werten arbeiten wollen. Glücklicherweise können diese Vorgänge mit wenigen einfachen Regeln bewältigt werden, wenn die Grundlagen und das System der Algorithmen einmal verstanden sind.

Lösungen

1) Das folgende Programm kehrt die Reihenfolge der in LABL1 gespeicherten Zeichenkette um:

6502

```
;
ORIGIN  ORG   $7000
LAST1   EQU   $0D
LABL1   DB    'THIS IS A MESSAGE'
TERMN8  DB    LAST1
;
BEGIN   LDX   #$FF
        LDA   #LAST1
        PHA
LOOP0   INX
        LDA   LABL1,X
        PHA
        CMP   #LAST1
ENDLP0  BNE   LOOP0
CLRSTK  PLA
;
BEGIN1  LDX   #$FF
LOOP1   INX
        PLA
        STA   LABL1,X
        CMP   #LAST1
ENDLP1  BNE   LOOP1
        RTS
```

In der Version des 6502 setzt der Code zwischen LOOP0 und ENDLOOP0 in einer Schleife die X-indizierte Adressierung ein, um die Zeichen nacheinander von LALB1 zu laden und auf den Stack zu schieben, nachdem der ASCII-Wert der Endkennung zuvor die Untergrenze des Stacks markiert hatte. Das letzte auf den Stack geschobene Zeichen ist ebenfalls eine Endkennung. Diese wird jedoch durch ihre Position als letztes Zeichen des Strings erzeugt. Damit ist die Schleife abgeschlossen, und das Zeichen mit der Endkennung auf der Spitze des Stacks wird mit CLRSTK gelöscht.
In der Z80-Version wird IX in der indirekten Adressierung eingesetzt, um den Akkumulator von LABL1 an zu laden. Die Zeichen des Strings LABL1 wechseln sich daher auf dem Stack mit den fortlaufenden Werten des Prozessor-Status-Registers ab.

Z80

```
        ORG   $C000
LAST1   EQU   $0D
LABL1   DB    'THIS IS A MESSAGE
TERMN8  DB    LAST1
;
BEGIN   LD    IX,LABL1-1
        LD    A,LAST1
        PUSH  AF
LOOP0   INC   IX
        LD    A,(IX+0)
        PUSH  AF
        CP    LAST1
ENDLP0  JR    NZ,LOOP0
CLRSTK  POP   AF
;
BEGIN1  LD    IX,LABL1-1
LOOP1   INC   IX
        POP   AF
        LD    (IX+0),A
        CP    LAST1
ENDLP1  JR    NZ,LOOP1
        RET
```

In beiden Versionen baut der Code zwischen BEGIN1 und ENDLP1 auf Techniken auf, die schon in früheren Programmen eingesetzt wurden. Diesmal wird jedoch der String in umgekehrter Folge von dem Stack gezogen und von LABL1 an aufwärts gespeichert. Die Schleife ist beendet, wenn das Zeichen der Endkennung auf dem Boden des Stacks gefunden wird.

Es ist wichtig, die Befehle für das Laden und Lesen des Stacks gegeneinander auszugleichen. Die Extrembedingungen sind das größte Problem. Wie sollen Anfang und Ende der Schleife aussehen, und welche „Aufräumarbeiten" (falls überhaupt) sind notwendig?

Der Z80-Befehl bei BEGIN und BEGIN1 (LD IX, LABL1−1) macht deutlich, wie nützlich ein Assembler-Programm sein kann. Der Ausdruck (LABL1−1) bedeutet hier „die Adresse des Bytes unmittelbar vor dem Byte, dessen Adresse LABL1 ist". Der Assembler übersetzt diese Anweisung automatisch in eine Adresse des Maschinencodes. Die meisten Assembler können Ausdrücke verarbeiten. Normalerweise lassen sich dabei ein oder zwei Operanden durch einfache arithmetische Zeichen ("+" oder "−") verändern.

2) Dieses Programm kehrt die Reihenfolge der Zeichen um, die in dem bei LABL1 abgelegten String enthalten sind. Die Anordnung

6502

```
;
ORIGIN   ORG   $7000
LAST1    EQU   $0D
SPACE    EQU   $20
LABL1    DB    'THIS IS'
TERMN8   DB    LAST1
;
BEGIN    LDX   #$FF
LOOPO    JSR   RVSWRD
         CMP   #LAST1
ENDLPO   BNE   LOOPO
         RTS
;
;****REVERSE A WORD S/R****
LASTCH   DB    $00
LASTX    DB    $00
RVSWRD   TXA
         TAY
         INY
RVSLPO   INX
         LDA   LABL1,X
         PHA
         CMP   #SPACE
         BEQ   CLRSTK
         CMP   #LAST1
ENDRVO   BNE   RVSLPO
CLRSTK   PLA
         STA   LASTCH
         STX   LASTX
RVSLP1   PLA
         STA   LABL1,Y
         INY
         CPY   LASTX
ENDLP1   BNE   RVSLP1
         LDA   LASTCH
         RTS
```

der Wörter bleibt dabei erhalten. Interessant ist hierbei, wie die Befehle JSR und CALL eingesetzt werden. Die Unterroutine RVSWRD hat eine ähnliche Struktur wie das Programm der Übung 1, kehrt jedoch nur die Zeichen eines Wortes und nicht den gesamten String um. In beiden Versionen (6502 und Z80) wird das Index-Register (X und IX) eingesetzt, um die Anfangsadresse des Wortes an die Unterroutine weiterzugeben. Dabei übergibt der Akkumulator den Wert des Zeichens an das Hauptprogramm, das den Ablauf beendet (entweder ein Leerzeichen oder die Endkennung).

Ein weiteres wichtiges Merkmal ist der Gebrauch des Y-Registers in der 6502-Version. Das Register enthält zunächst die Anfangsadresse des Wortes, während X als Index für die Stack-Schleife dient. Danach wird Y als Index der Schleife eingesetzt, die das Herunterziehen vom Stack ausführt, während X die Endadresse des Wortes enthält. Der Begriff „Adresse" ist hier etwas ungenau, da X und Y Ein-Byte-Register sind und keine vollständige Adresse enthalten können. Statt dessen speichern sie einen Offset zu der Adresse LABL1.

In der Z80-Version werden statt IX und IY die Registerpaare HL und DE eingesetzt. Wie die X- und Y-Register des 6502 speichern sie die Anfangs- und Endadressen des Wortes, werden aber statt als Indizes über eine Basisadresse als indirekte Adressen eingesetzt

Z80

```
         ORG   $C000
LAST1    EQU   $0D
SPACE    EQU   $20
LABL1    DB    'THIS IS A MESSAGE'
TERMN8   DB    LAST1
;
BEGIN    LD    DE,LABL1-1
LOOP0    CALL  RVSWRD
         CP    LAST1
ENDLP0   JR    NZ,LOOP0
         RET
;
;***REVERSE A WORD S/R***
LASTCH   DB    $00
RVSWRD   PUSH  DE
         POP   HL
         INC   HL
RVSLP0   INC   DE
         LD    A,(DE)
         PUSH  AF
         CP    SPACE
         JR    Z,CLRSTK
         CP    LAST1
ENDRV0   JR    NZ,RVSLP0
CLRSTK   POP   AF
         LD    (LASTCH),A
;
RVSLP1   POP   AF
         LD    (HL),A
         INC   HL
         LD    A,L
         CP    E
         JR    NZ,RVSLP1
         LD    A,H
         CP    D
ENDRV1   JR    NZ,RVSLP1
         LD    A,(LASTCH)
         RET
```

(der Befehl LD A, (DE) bedeutet „Lade den Akkumulator aus dem Byte, dessen Adresse in DE gespeichert ist"). Alle Registerpaare des Z80 lassen sich auf diese Art verwenden. Da dem Befehlssatz jedoch im Zwei-Byte-Format jegliche Vergleichsmöglichkeit fehlt, müssen bei einem Vergleich der Inhalte von DE und HL die Register E mit L und D mit H einzeln verglichen werden. Da der 6502 keinen Befehl für den unmittelbaren Vergleich von X und Y hat, können hier X und Y nur mit Hilfe einer Speicherstelle indirekt miteinander verglichen werden.

Alles Schiebung

In dieser Folge sehen wir uns die Probleme der Subtraktion und Multiplikation genauer an und stellen eine neue Klasse von Befehlen vor: die Op-codes für Shift und Rotate.

Der Z80 und der 6502 unterstützen beide den SBC-Befehl (Subtrahieren mit Übertrag), führen ihn jedoch völlig verschieden aus. Der 6502 benutzt das Übertragsflag, um ein „Borgen" anzuzeigen, das dem Übertrag der Addition entspricht. In der Assemblersprache des Z80 funktioniert SBC genau wie der Befehl ADC, wobei das Übertragsflag jedoch je nach Ergebnis gesetzt oder auf Null gestellt wird.

Wenn Sie mit ADC $E4 auf $5F addieren (nachdem das Übertragsflag zuvor auf Null gesetzt wurde), erscheint im Akkumulator die Summe $43 mit gesetztem Flag; das vollständige Ergebnis ist daher $0143. Das Übertragsflag wurde gesetzt, da der Akkumulator allein das vollständige Ergebnis nicht speichern kann.

Wenn Sie auf dem Z80 nun wiederum das Übertragsflag löschen und dann $E4 von $5F subtrahieren, steht im Akkumulator als Ergebnis $7B mit gesetztem Flag. Bei einer erneuten Addition von $7B auf $E4 (wiederum nachdem das Übertragsflag gelöscht wurde) enthält der Akkumulator als Ergebnis $5F mit gesetztem Flag. Die folgenden Zeilen zeigen, wie der Vorgang abläuft:

```
$5F — $E4 = $7B Übertragsflag gesetzt
$5F = $E4 + $7B Übertragsflag gesetzt
```

Wenn Sie das Flag als Indikator eines negativen Ergebnisses nehmen, läßt sich $7B als Zweierkomplement verstehen:

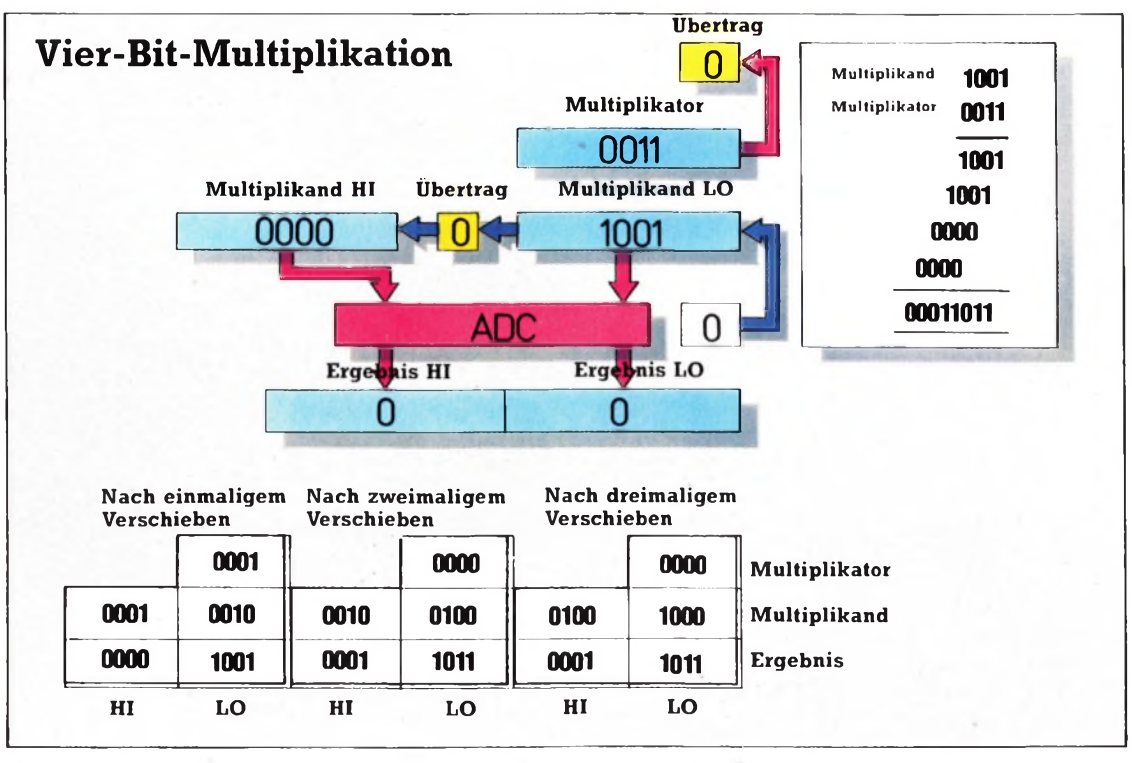
```
$7B binär                 = 01111011
minus Eins                  —1
ergibt das
Einerkomplement      <-   01111011
      Umkehrung
ergibt das
Zweierkomplement          10000100 = $85
```

$5F – $E4 sollte daher die negative Zahl –$85 ergeben. Überprüfen Sie das Ergebnis im Dezimalsystem:

```
  $5F =    95 dezimal
 —$E4 =   228 dezimal
  $85 = —133 dezimal
```

So weit, so gut. Nehmen Sie nun an, daß die Subtraktion eigentlich eine Zwei-Byte-Zahl betraf: $375F – $21E4.

```
 HI  LO
 $37 5F =  14175 dezimal
—$21 E4 = — 8676 dezimal
 $15 7B =   5499 dezimal
```



Das Beispiel zeigt aus Gründen der Einfachheit eine Vier-Bit-Multiplikation – die Anzahl der Bits hat keinen Einfluß auf den Algorithmus. Dabei wird deutlich, wie der Inhalt der einzelnen Bits des Multiplikanden – Eins oder Null – das Ergebnis aus der Addition von Nullen oder den verschobenen Versionen des Multiplikanden entstehen lassen. Die Bits des Multiplikators werden nach rechts über das Übertragsflag verschoben, während die Multiplikandenbits durch eine Linksrotation vom niederwertigen Byte durch das Übertragsflag in das höherwertige Byte wandern.

Bei der Subtraktion des niederwertigen Bytes ist das Ergebnis $7B mit gesetztem Übertragsflag. Dieser Übertrag wird dann von dem SBC-Befehl auf die Zahl $21 addiert, die dadurch $22 wird. Von $37 subtrahiert ergibt sich $15.

Die Zwei-Byte-Arithmetik des Z80 besteht daher aus diesem einfachen Ablauf:
1) Übertragsflag löschen.
2) Die niederwertigen Bytes mit Übertrag subtrahieren.
3) Die höherwertigen Bytes mit Übertrag subtrahieren.

Die Version des 6502 arbeitet schon im ersten Teilvorgang anders: Das Übertragsflag muß gesetzt sein, damit die niederwertigen Bytes von den höherwertigen Bytes „borgen" können. Ist kein „Borgen" notwendig, läuft die Subtraktion normal ab, wobei das Übertragsflag für die Subtraktion der höherwertigen Bytes bestehen bleibt. Tritt in der Subtraktion des niederwertigen Bytes jedoch ein Unterlauf auf, übernimmt das Übertragsflag die Funktion des „neunten" Akkumulatorbytes. Nachdem auf diese Weise sichergestellt wurde, daß das richtige Ergebnis auftritt, wird das Flag gelöscht. Auf dem 6502 erzeugt die Subtraktion höherwertiger Bytes mit gelöschtem Übertragsflag das gleiche Ergebnis wie eine Z80-Subtraktion mit gesetztem Übertragsbyte, da die Zahl, die subtrahiert werden soll, vor dem Subtraktionsvorgang dekrementiert wird. Beide Methoden, einen Unterlauf abzufangen, gehen auf die alten mathematischen Methoden des „Borgens" und „Übertragens" zurück. Doch sehen wir uns die Version des 6502 einmal genauer an.

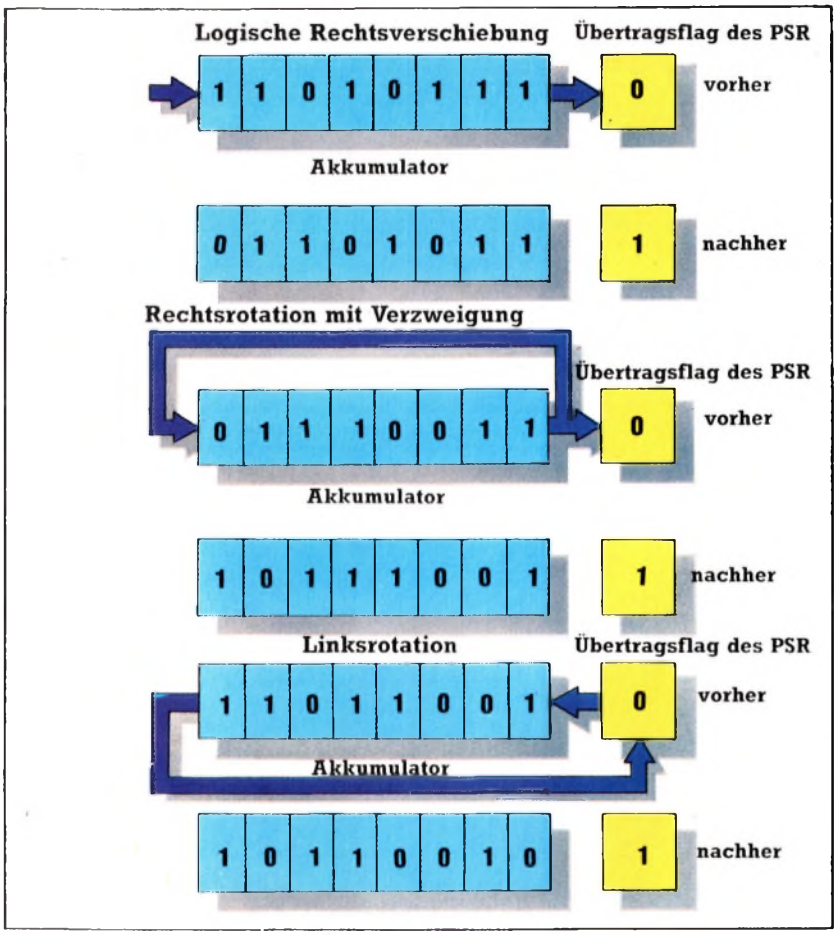
Wenn das Übertragsflag gelöscht ist und wir $E4 von $5F subtrahiert haben, steht im Akkumulator als Ergebnis $7A, wobei das Flag nicht gesetzt ist. Das Beispiel des Z80 zeigte, daß die Berechnung $7B ergab, wobei das gesetzte Übertragsflag eine negative Zahl anzeigte. $7B ist das Zweierkomplement des „echten" Ergebnisses (-$85). $7A muß daher das Einerkomplement dieser Zahl sein und der Status des Übertragsflags eine Art Schalter des Akkumulators. Bei gesetztem Flag ergibt sich das Zweierkomplement, ist es nicht gesetzt, das Einerkomplement.

Wird die Subtraktion auf dem 6502 mit gesetztem Übertragsflag durchgeführt, enthält der Akkumulator $7B, und das Flag ist dadurch gelöscht.

Multiplikationen

Untersuchen wir eine normale dezimale Multiplikation:

```
 174   Multiplikand
*209   Multiplikator
 348   Erstes Teilergebnis
 000   Zweites Teilergebnis
1566   Drittes Teilergebnis
36366  Endergebnis
```



Der Ablauf läßt sich leicht nachvollziehen, ohne daß Sie verstehen müssen, auf welchen mathematischen Prinzipien die Schreibweise mit unterschiedlichen Positionen beruht. Die Methode besteht darin, jedes Teilergebnis gegenüber dem vorigen Teilergebnis um genau eine Stelle nach rechts zu rücken (damit der Vorgang anschaulicher wird, wurden die leeren Stellen nicht mit Nullen aufgefüllt).

Viele Menschen haben Schwierigkeiten mit dem Zusammenspiel der versetzten Teilergebnisse und dem Einmaleins. Die binäre Multiplikation ist einfacher. Sie kennt nur einen Rechenvorgang: Eins mal Eins. Alle anderen Ein-Byte-Ergebnisse sind Null. Sehen Sie sich folgende binäre Multiplikation an:

```
 1101    = dezimal 13
*1001    = dezimal 9
 1101      Erstes Teilergebnis
 0000      Zweites Teilergebnis
 0000      Drittes Teilergebnis
 1101      Viertes Teilergebnis
1110101 = 117 dezimal
```

In diesem Beispiel wird nicht nur die Verschiebung der Teilergebnisse deutlich, sondern auch die bestechende Einfachheit der binären Multiplikation. Ein Teilergebnis kann nur Null oder der verschobene Multiplikand sein, je nachdem ob das entsprechende Bit des Multiplikators Eins oder Null ist. Diese Struktur ist

Die Befehle für Verschieben und Rotation werden hauptsächlich für die bitweise Untersuchung eines Registers eingesetzt. Bei jeder Rotation wird das höchste oder das niedrigste Bit in das Übertragsflag des PSR gestellt. Der Status des Übertragsflags kann dann von einem Verzweigungsbefehl abgefragt werden und so den Programmablauf steuern. Bei den Rotationsbefehlen bleibt der Inhalt des Registers erhalten, während die logische Verschiebung die leeren Stellen mit Nullen füllt. Eine Linksverschiebung multipliziert die Register daher mit zwei, während die Rechtsverschiebung sie durch zwei dividiert.

Acht-Bit-Multiplikation

6502			Z80		
	ORG	$C100		ORG	$D000
START	LDA	#$00	START	LD	BC,(MPR)
	STA	PRODLO		LD	B,$08
	STA	PRODHI		LD	DE,(MPDLO)
	STA	MPDHI		LD	D,$00
	LDX	#8		LD	HL,$00
	CLC		LOOP0	SRL	C
LOOP0	ROR	MPR		JR	NC,CONT0
	BCC	CONT0		CALL	ADDIT
	JSR	ADDIT	CONT0	SLA	E
CONT0	ASL	MPDLO	ENDLP0	DJNZ	LOOP0
	ROL	MPDHI		LD	PRODLO
	DEX			RTS	
ENDLP0	BNE	LOOP0	MPR	DB	$E2
	RTS		MPDLO	DB	$7A
MPR	DB	$E2	MPDHI	DB	$00
MPDLO	DB	$7A	PRODLO	DW	$0000
MPDHI	DB	$00	ADDIT	ADD	HL,DE
PRODLO	DB	$00		RET	
PRODHI	DB	$00			
ADDIT	CLC				
	LDA	PRODLO			
	ADC	MPDLO			
	STA	PRODLO			
	LDA	PRODHI			
	ADC	MPDHI			
	STA	PRODHI			
	RTS				

den Tests ähnlich, denen wir in den Steuermechanismen der Assemblersprache schon begegnet sind. In der binären Multiplikation muß nacheinander jedes Bit des Multiplikators untersucht werden – ist es Null, wird Null addiert; ist es Eins, wird der versetzte Multiplikand auf die Gesamtsumme addiert. Wie können nun die einzelnen Bits des Multiplikators untersucht werden, und wie läßt sich der Multiplikand versetzen?

Der Zustand eines bestimmten Bits innerhalb eines Bytes läßt sich mit dem Befehl BIT feststellen, den es im Instruktionssatz beider Prozessoren gibt. Auf dem Z80 enthält dieser Befehl eine Adresse und eine Bitzahl als Operand. Dabei wird das Nullflag gesetzt, wenn das entsprechende Bit Null ist, und gelöscht, wenn das Bit Eins ist. Der BIT-Befehl des 6502 enthält nur die Adresse. Der Adresseninhalt wird über AND mit dem Akkumulator verknüpft, und das Nullflag wird gesetzt oder nicht gesetzt, je nachdem ob das Ergebnis wahr oder falsch ist.

Die Instruktionssätze des Z80 und des 6502 besitzen dafür eine Reihe von Verschiebungs- und Rotationsbefehlen, wobei die des Z80 komplexer sind. Generell verschieben sie jedes Registerbit um eine Position nach rechts oder links. Die Unterschiede der Prozessoren liegen in der Behandlung des letzten Registerbits – das Bit, das sich aus dem Register heraus bewegt, während am anderen Ende ein Bit eingefügt wird. Die Verschiebung von Bit 7 aus dem Register und das Einsetzen in Bit 0 nennt man Linksrotation. Wird Bit 0 auf Bit 7 verschoben, spricht man von einer Rechtsrotation. Der übrige Inhalt des Registers wird dabei entsprechend verändert, wobei jedoch keine neuen Werte eingefügt werden. Nach acht Rotationen ist der ursprüngliche Zustand des Registers wiederhergestellt.

Ohne Rotation sind eine Bestimmungsadresse für das herausgeschobene Bit und eine Quelle für das eingesetzte Bit notwendig. Beide Funktionen werden oft von den Flags des Prozessor-Status-Registers und speziell von dem Übertragsflag übernommen. Für den Aufbau einer Multiplikationsroutine für zwei Ein-Byte-Zahlen muß der Multiplikand nach links und der Multiplikator nach rechts verschoben werden. Dabei werden die Bits des Multiplikanden in das höherwertige Multiplikandenbyte verschoben und die unbelegten Bits mit Nullen gefüllt. Die Multiplikatorbits müssen dann zum Testen über die PSR-Flags geleitet werden. Den Zustand dieser und der neu eingeschobenen Multiplikatorbits braucht man nur dann zu beachten, wenn sie nochmals gebraucht werden. Bei dem Multiplikator ist für uns nur wichtig, ob das ausgeschobene Bit Eins oder Null ist.

In der linksstehenden Abbildung wurde der Multiplikator in der Adresse MPR untergebracht, der Multiplikand in MPDLO und das Ergebnis in PRODLO und PRODHI.

In diesem Beispiel wird deutlich, wieviel einfacher die 16-Bit-Register und der zugehörige Befehlssatz die Programmierung des Z80 gestalten. Sehen Sie sich dafür die ADDIT-Unterroutine beider Programme an. Der 6502 rotiert den Multiplikator mit ROR bitweise nach rechts in den Übertrag und benutzt ASL und ROL, um den Multiplikanden nach links aus MPDLO nach MPDHI und in den Übertrag zu schieben. Die Schleife wird von dem X-Register gesteuert, das als Zähler dient.

Die Z80-Version verschiebt den Multiplikator mit SRL nach rechts in den Übertrag und den Multiplikanden nach links über den Übertrag in das Registerpaar DE. Das Register B ist der Schleifenzähler. Interessant ist hierbei, daß der ADD-Befehl nicht nur eine 16-Bit-Arithmetik unterstützt, sondern anders als ADC nicht von dem Übertragsflag beeinflußt wird.

In der nächsten Folge untersuchen wir die Division und die Techniken, mit der die Bildschirmdarstellung gesteuert werden kann.

Übung

1) Schreiben Sie eine Unterroutine für die Multiplikation eines 16-Bit-Multiplikanden mit einem Acht-Bit-Multiplikator.

2) Die Multiplikation ist nur eine wiederholte Addition. Schreiben Sie eine Routine für eine Acht-Bit-mal-acht-Bit-Multiplikation, die keine Befehle für Verschiebung/Rotation enthält.

Geteilte Zahlen

Wir beenden die allgemeine Einführung in den Maschinencode mit einem kurzen Blick auf die binäre Division und sehen uns an, wie der Bildschirm mit Maschinensprache gesteuert werden kann.

Ebenso wie uns die Handmultiplikation den Algorithmus für die binäre Multiplikation liefert, so läßt sich auch die Handdivision als Modell für die binäre Division einsetzen:

```
00001110 Rest 00          Quotient
10011010:1011             Dividend mit Teiler
-1011                     subtrahiere Teiler
 10000
 -1011                    subtrahiere Teiler
  1011
  -1011                   subtrahiere Teiler
     00                   kein Rest
```

Die Methode beruht auf der wiederholten Subtraktion des Teilers von den hochwertigen Bits des Dividenden. Je nach Ergebnis der Subtraktion wird dabei eine Null oder eine Eins in den Quotienten eingesetzt. Die letzte Subtraktion des Teilers liefert den Rest.

Dieser Algorithmus läßt sich auf verschiedene Weise programmieren. Der Ablauf ist jedoch längst nicht so durchsichtig wie bei der Multiplikation. Dem Z80 stehen hier wiederum die Schnelligkeit und Flexibilität seiner 16-Bit-Register zur Verfügung, während der 6502 nur acht Bits gleichzeitig bearbeiten kann. Der Teiler (Divisor) wird in der Adresse DIVSR untergebracht, der Dividend in DVDND, der Quotient in QUOT und der Rest in RMNDR.

Rotationen

Wenn die Subtraktion des Teilers vom Teildividend ein negatives Ergebnis liefert, muß der Dividend (durch eine Addition des Teilers) wieder in seinen ursprünglichen Zustand versetzt werden. Beachten Sie in der Version des 6502 die Behandlung des Prozessor-Status-Registers: Das Übertragsflag muß in den Quotient rotiert werden, dabei jedoch seinen Zustand beibehalten, um das Ergebnis der Subtraktion anzeigen zu können. Das Programm stellt den ursprünglichen Status des Übertrags her, indem es das PSR vor der Rotation auf den Stack schiebt und unmittelbar danach wieder zurücklädt.

Mit der Umsetzung der vier Grundrechenarten in den Maschinencode gewannen wir einen tieferen Einblick in die Mechanik der Maschinenvorgänge. Es ist jedoch nicht notwendig, alle möglichen arithmetischen Vorgänge mit einem oder mehreren Bytes aufzuzeigen, da diese Routinen bereits in zahlreichen Lehrbüchern veröffentlicht wurden. Sollte es jedoch notwendig werden, die bekannten Routinen zu verändern, dann werden wir die Änderungen abdrucken oder als „Hausaufgabe" stellen.

Bis jetzt haben wir den Arbeitsspeicher und die CPU als Rechenmaschine behandelt, deren Ergebnisse wir uns mit einem Monitorprogramm im RAM angesehen haben. Selbstverständlich genügt diese Anzeigemethode nicht. Es hatte jedoch wenig Sinn, vor Abschluß der arithmetischen Mechanik und dem Aufruf von Unterroutinen auf die Anzeigemöglichkeiten des Maschinencodes einzugehen.

Numerierung der Pixelbytes

Die Anzeigen der meisten Microcomputer arbeiten mit Memory-Maps (bestimmte RAM-Bereiche enthalten Abbilder des Bildschirms). Die Anzeige setzt sich aus Pixeln zusammen, die entweder ein- oder ausgeschaltet sind und von binären Einsen (An) oder Nullen (Aus) dargestellt werden. Der gesamte Inhalt des Bildschirms findet sich daher in den Bits des Bildschirmspeichers wieder. Obwohl der Acorn B, der Spectrum und der Commodore diese Methode verwenden, setzen alle drei Geräte andere Versionen ein. Für uns würde es genügen, jede Bildschirmzeile in Pixelbytes aufzuteilen, die von links nach rechts fortlaufend numeriert wären. Dabei würde das links außen stehende Byte einer Zeile immer dem rechts außen stehenden Byte der darüberliegenden Zeile folgen. Da keine der Maschinen mit dieser einfachen Lösung arbeitet, müssen wir jedes Gerät gesondert behandeln.

Die Anzeige des Spectrum ist ständig auf hohe Auflösung eingestellt, wobei dem Bildschirminhalt ein fester Speicherbereich zugeordnet ist. Die Punkte sind jedoch kompliziert angeordnet: Der Bildschirm wird horizontal in drei Blöcke mit je acht Zeilen unterteilt, wobei jede Zeile wiederum acht Pixelzeilen enthält. Innerhalb der Zeilen werden die Bytes sequentiell adressiert, die Zeilen selbst stehen jedoch nicht hintereinander. Der Acorn B und der Commodore 64 haben eine andere Anordnung, die jedoch auf einem ähnlichen Konzept beruht. Wir beschränken uns deshalb darauf, einzelne ASCII-Zeichen auf den Bildschirm zu bringen.

Im ROM befinden sich einige Maschinencoderoutinen für die Steuerung der Anzeige. Mit einer genauen Beschreibung dieser Routinen lassen sie sich von der Assemblersprache aus aufrufen. Für diesen Vorgang muß man jedoch die Anfangsadresse, die eingesetzten Register sowie alle notwendigen Vorbedingungen genau kennen.

Auf dem Spectrum brauchen keine Vorbedingungen erfüllt zu werden. Als Register wird der Akkumulator verwandt, der den ASCII-Code des darzustellenden Zeichens enthalten muß. Bei der Ausgabe des Befehls RST $10 wird das Zeichen, dessen Code im Akkumulator gespeichert ist, an der augenblicklichen Cursorposition angezeigt. Auch die beiden anderen Systeme arbeiten nach dieser Methode. Den Op-code RST (ReSTart) gibt es jedoch nur im Befehlssatz des Z80. Er ist ein Ein-Byte-Befehl, der die Zero-Page anspricht und nur einen von acht möglichen Operanden annehmen kann – $00, $08, $10, $18 bis $38. Jede dieser Speicherstellen zeigt auf die Anfangsadresse einer ROM-Routine auf der Zero-Page, die die Ein- und Ausgabe steuert. Da RST schneller ist (RST läuft mit größerer Geschwindigkeit ab als CALL – allerdings würde nur die CPU den Unterschied merken) und Programme sich dadurch auf andere Geräte übertragen lassen, werden diese Routinen über den RST-Befehl aufgerufen und nicht durch ihre direkten Anfangsadressen. Wenn allen Z80-Programmierern bekannt ist, daß auf Z80-Maschinen der Befehl RST $10 die PRINT-Routine aufruft, dann braucht sich niemand Gedanken darüber zu machen, wo die Konstrukteure sie untergebracht haben. Die Konstrukteure ihrerseits können sie an jede beliebige Position setzen, wenn die RST-Speicherstellen auf die richtigen Anfangsadressen zeigen.

Auf dem Acorn B läuft der Vorgang ähnlich ab: Bei Aufruf des Befehls JSR $FFEE wird der im Akkumulator gespeicherte ASCII-Code an der augenblicklichen Cursorposition des Bildschirms angezeigt. Bei $FFEE beginnt die OSWRCH-Routine, die in der Dokumentation des Acorn B häufig erwähnt wird.

Übertragung der Daten

Auch der Commodore 64 folgt diesem Muster. Mit dem Befehl JSR $FFD2 wird der im Akkumulator gespeicherte ASCII-Code an der augenblicklichen Position des Cursors als Zeichen auf den Schirm gebracht. Die entsprechende CHKOUT-Routine ist im Handbuch beschrieben.

Die Abläufe zeigen, wie sich ROM-Routinen einsetzen lassen und wie Kommunikationsregister funktionieren. Die Datenübertragung zwischen dem aufrufenden Programm und der Unterroutine funktioniert in beiden Richtungen. So kann eine Eingaberoutine zum Beispiel die Zeichen eines Peripheriegerätes über den Akkumulator an die CPU übermitteln. Selbst wenn dabei keine gültigen Daten übertragen werden, so kann die Unterroutine über eines dieser Register zumindest eine Fehlermeldung ausgeben. Auf Tastatureingaben und die anderer Peripheriegeräte werden wir in späteren Folgen eingehen, ebenso wie auf das Plotten mit hoher Auflösung.

16-Bit- und 8-Bit-Division

Z80			6502		
START	LD	A,(DIVSR)	START	LDA	#$00
	LD	D,A		STA	QUOT
	LD	E,$00		STA	RMNDR
	LD	HL,(DVDND)		LDX	#$08
	LD	B,$08		LDA	DVDHI
LOOP0	AND	A		SEC	
	SBC	HL,DE		SBC	DIVSR
	INC	HL	LOOP0	PHP	
	JP	P,POSRES		ROL	QUOT
NEGRES	ADD	HL,DE		ASL	DVDLO
	DEC	HL		ROL	A
POSRES	ADD	HL,HL		PLP	
	DJNZ	LOOP0		BCC	CONT1
	LD	(QUOT),HL		SBC	DIVSR
	RET			JMP	CONT2
DIVSR	DB	$F9	CONT1	ADC	DIVSR
DVDND	DW	$FDE8	CONT2	DEX	
QUOT	DB	$00		BNE	LOOP0
RMNDR	DB	$00		BCS	EXIT
				ADC	DIVSR
				CLC	
			EXIT	ROL	QUOT
				STA	RMNDR
				RTS	
			DIVSR	DB	$F9
			DVDLO	DB	$E8
			DVDHI	DB	$FD
			QUOT	DB	$00
			RMNDR	DB	$00

Der Maschinensprachkurs begann mit einem generellen Überblick. Sie haben gesehen, wie sich die Bytes des RAM als Folge von ASCII-Zeichen interpretieren lassen oder als aneinandergereihte Zwei-Byte-Adressen oder auch als Befehlssequenz des Maschinencodes. Der Maschinencode-Monitor zeigte, daß sich die gleichen Bytes als drei völlig verschiedene, aber dennoch gültige Befehlssequenzen interpretieren lassen – abhängig davon, ob die Disassemblierung beim ersten, zweiten oder dritten Byte der Folge anfängt.

Danach wurde genauer auf Speicherorganisation und Adreßformate eingegangen. Schon kurze Zeit, nachdem Sie sich mit binärer Arithmetik beschäftigt hatten, konnte festgestellt werden, welchen Begrenzungen die CPU unterworfen ist: Acht-Bit-Prozessoren sind – bis auf einige Ausnahmen – an die Grenzen eines Bytes gebunden (anders ausgedrückt: an den Dezimalbereich von 0 bis 255). Mit der Einführung der binären Arithmetik wurde deutlich, wie unhandlich in diesem Umfeld das Dezimalsystem ist. Die Vorstellung der Spei-

cherseiten zeigte, wie die logische Größe der Seiten aus der Zahlenbasis entsteht, und machte deutlich, daß in einem binären System die Grundlage immer ein Vielfaches von Zwei sein muß. Sie haben in der Zahl 256 (zwei hoch acht) die magische Zahl des Systems der Acht-Bit-Prozessoren erkannt.

Da sich auch das Binärsystem sehr schnell als unhandlich und fehleranfällig erwies, gingen wir auf das Hexadezimalsystem (Zahlenbasis 16) und die dazugehörige Mathematik über. Sie sahen, wie ein Acht-Bit-Byte durch zwei Hexadezimalstellen (von $00 bis $FF) dargestellt werden kann, wobei eine Hexstelle jeweils die vier niederwertigen Bits anzeigt und die andere die vier höherwertigen Bits eines Bytes.

Zusammenfassung

Auf dieser Grundlage wurde untersucht, wie BASIC-Programme im RAM-Bereich gespeichert sind. Mit der Beschreibung der Token erhielten Sie Einblick in die Mechanik der Betriebssysteme, während die Zeilenendmarkierung deutlich machte, wie ein BASIC-Interpreter das Ende einer Zeile erkennen kann. Die Link-Adressierung des Commodore führte schließlich in die lo-hi-Adreßkonvention und in die indirekte Adressierung ein.

Von diesem Punkt an beschäftigten wir uns mit der Assemblersprache. Dabei wurde zunächst untersucht, wie die CPU mit Hilfe von Acht-Bit-Op-codes einfache Aufgaben ausführt. Von diesem Konzept war es nur ein kleiner Schritt zu den mnemotischen Kürzeln der Assemblersprache. Es wurde deutlich, daß die Programmierung im Maschinencode, in der Assemblersprache oder in BASIC nur unterschiedliche Methoden sind, Abläufe festzulegen. Dabei ist die logische Lösung eines Problems wesentlich wichtiger als die eigentliche Umsetzung in den Programmcode.

In den nächsten Folgen beschäftigten wir uns dann mit dem Laden und Starten des Maschinencodes mit Hilfe von BASIC-Programmen. Dabei sahen wir uns auf dem Acorn B, Spectrum und Commodore 64 die Systemvariablen und Pointer der Betriebssysteme genauer an und fanden heraus, wie man dem BASIC Speicherplatz „stehlen" kann.

Nach einem kurzen Blick auf kleine Computersysteme und auf die Zentraleinheiten Z80 und 6502 haben Sie Ihre ersten Assemblerprogramme geschrieben, die den Inhalt des Speichers und des Akkumulators veränderten. Mit den Assembleranweisungen oder Pseudo-Opcodes entfernten wir uns von den verwirrenden Einzelheiten des Maschinencodes und der Handassemblierung und kamen der komfortablen „realen" Assemblerprogrammierung einen großen Schritt näher.

An diesem Punkt gingen wir auf die logische Struktur der Programmiersprache und damit auf die Aufgabe des Prozessor-Status-Registers (PSR) ein. Seine Rolle als Anzeigeinstrument der CPU-Vorgänge wurde bei der Einführung binärer mathematischer Abläufe – und besonders bei dem Befehl „Addiere mit Übertrag" – deutlich. Von nun an drehte sich der Kurs hauptsächlich um das PSR und die damit eingesetzten Befehle.

Nachdem wir die verschiedenen Adressierarten gestreift hatten, lernten Sie die indizierte Adressierung ausführlicher kennen, da sie für Schleifen, Listen und Tabellen wichtig ist. Eine Steuerung des Programmflusses wurde notwendig. Daher untersuchten wir die Befehle der bedingten Verzweigung. Mit der Programmsteuerung, einfacher Arithmetik und matrixähnlichen Tabellen hatten wir nun fast das gesamte Grundgerüst einer Programmiersprache. Jetzt fehlten nur noch Übung und die systematische Untersuchung aller Einsatzmöglichkeiten.

Der Aufruf von Assembler-Unterroutinen und die Rücksprungtechnik führten uns in einen der letzten noch unerwähnten Bereiche des Betriebssystems: den Stack. Dieses Instrument zeigte Ihnen einige neue Kniffe der Maschinencodeprogrammierung. Dabei eröffnete eine genauere Untersuchung der CPU-Register und ihrer Anwendung weitere Möglichkeiten zur Steuerung von Speicher und Microprozessor.

Schließlich waren Sie mit dem Aufbau des Microprozessors vertraut und hatten auch genügend Op-code-Befehle zur Verfügung, um genauer auf die binäre Arithmetik einzugehen. Die Eigenheiten der Subtraktion und des Zweierkomplements und auch der Aufbau von Multiplikation und Division mit Verschieben und Rotation wurden untersucht.

In der Fortsetzung des Kurses werden wir uns mit der Programmierung im Maschinencode weiter vertraut machen und die beiden Prozessoren – Z80 und 6502 – für spezielle Aufgaben einsetzen. Dabei behandeln wir auch die 6809-CPU des Dragon 32 und 64.

Lösungen

1) Die schnellste Lösung ist sicherlich eine Routine, die speziell auf 16-Bit-Multiplikanden ausgerichtet ist – ähnlich wie die Acht-Bit-Routine der letzten Folge. Sie können aber auch eine 16-Bit-Multiplikation in zwei separate Acht-Bit-Multiplikationen aufteilen (zunächst mit dem niederwertigen Byte, dann mit dem höherwertigen Byte multiplizieren). Dabei wird die bestehende Acht-Bit-Routine zweimal aufgerufen, der Übertrag aus dem niederwertigen Byte berücksichtigt und das Resultat in den Ergebnisbytes gespeichert.

2) Eine Multiplikation durch fortgesetzte Addition läßt sich leicht über eine Schleife programmieren, deren Zähler als Multiplikator dient. Bei jedem Schleifendurchgang wird der Multiplikand auf das Ergebnis addiert.

ROR – Bit nach rechts rotieren — 6502

Register – 6A (1 Byte)

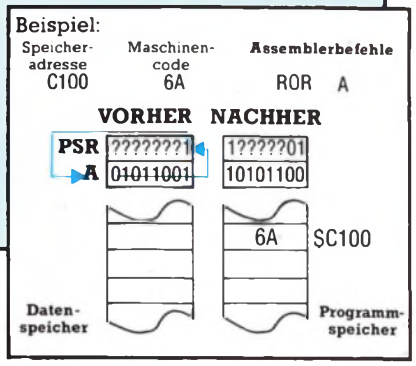
Der Inhalt des Bytes wird über den Übertrag um ein Bit nach rechts verschoben.

Bewirkt beim PSR:

	SV		BD	I	ZC
MSB	X				X X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	6A	ROR A



RR – Bit nach rechts rotieren — Z80

Register – CB (2 Bytes)

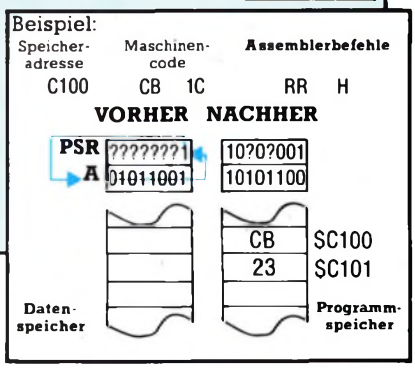
Der Inhalt des Bytes wird über den Übertrag um ein Bit nach rechts verschoben.

Bewirkt beim PSR:

	SZ	H	VNC
MSB	X X	0	X 0 X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	CB 1C	RR H



SBC – Subtrahieren mit Übertrag — 6502

Absolut – ED (3 Bytes)

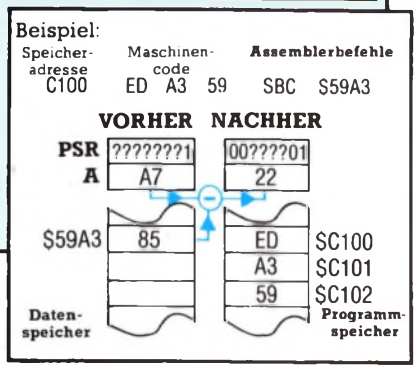
Der Inhalt der Speicherstelle wird vom Akku. subtrahiert. Der Übertrag zeigt den Status des Borgens.

Bewirkt beim PSR:

	SV		BD	I	ZC
MSB	X X				X X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	ED A3 59	SBC $59A3



SBC – Subtrahieren mit Übertrag — Z80

Konstante – DE (2 Bytes)

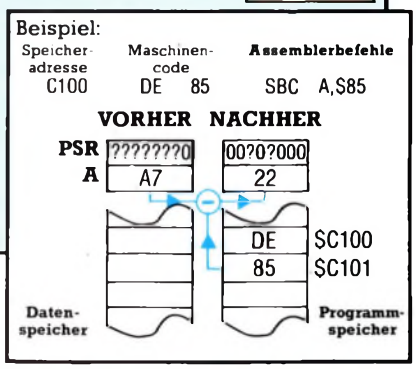
Der Inhalt der Speicherstelle, die dem Op-code folgt, wird vom Akku subtrahiert.

Bewirkt beim PSR:

	SZ	H	VNC
MSB	X X	X	X X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	DE 85	SBC A,$85



ASL – Arithm.nach links schieben — 6502

Register – OA (1 Byte)

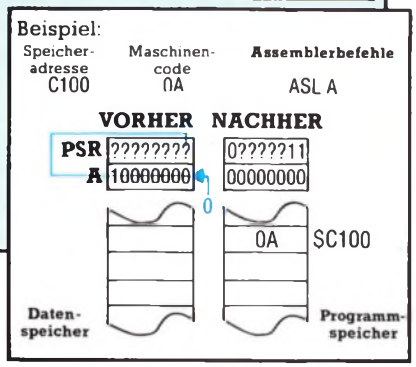
Der Inhalt des Bytes wird um ein Bit nach links in den Übertrag verschoben; das niederwertigste Bit wird Null.

Bewirkt beim PSR:

	SV		BD	I	ZC
MSB	X				X X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	0A	ASL A



SLA – Arithmetisch nach links schieben — Z80

Register – CB (2 Bytes)

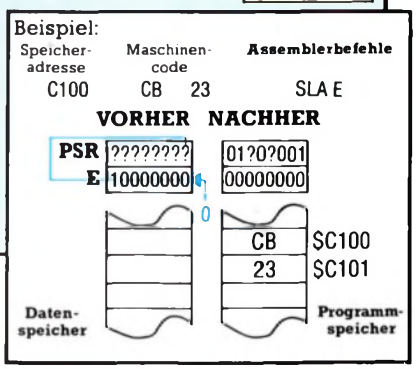
Der Inhalt des Bytes wird um ein Bit nach links in den Übertrag verschoben; das niederwertigste Bit wird Null.

Bewirkt beim PSR:

	SZ	H	VNC
MSB	X X	0	X 0 X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	CB 23	SLA E



CMP – Mit Akkumulator vergleichen — 6502

Absolut – CD (3 Bytes)

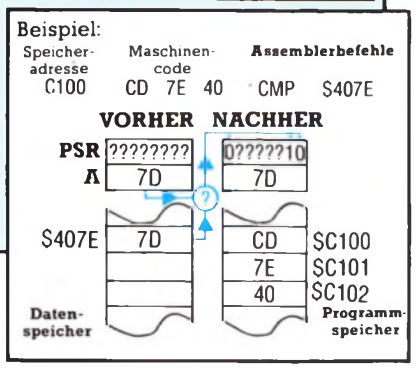
Der Inhalt der Speicherstelle wird mit dem Akkumulator verglichen.

Bewirkt beim PSR:

	SV		BD	I	ZC
MSB	X				X X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	CD 7E 40	CMP $407E



CP – Mit Akkumulator vergleichen — Z80

Konstante – FE (2 Bytes)

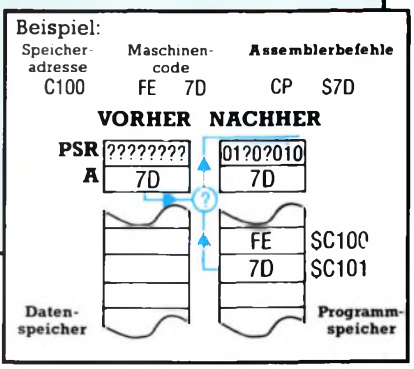
Der Inhalt der Speicherstelle, die dem Op-code folgt, wird mit dem Akku verglichen.

Bewirkt beim PSR:

	SZ	H	VNC
MSB	X X	X	X 1 X LSB

Beispiel:

Speicheradresse	Maschinencode	Assemblerbefehle
C100	FE 7D	CP $7D



Fenstertechnik

Wir stellen eine Maschinencoderoutine vor, die auf dem Spectrum die Programmierung von Fenstern möglich macht. Ein Beispielprogramm in BASIC zeigt, wie der Text innerhalb der Fenster bewegt wird.

Das Wort „HELLO" kann in Pixelschritten horizontal und vertikal über das Fenster bewegt werden.

Mit unserem Maschinencodeprogramm können Sie auf dem Spectrum rechteckige Bildschirmfenster definieren, deren Inhalt sich auf und ab und nach rechts und links bewegen läßt. Die Größe der Fenster ist beliebig, sie können an jeder Position des Bildschirms stehen und sind nicht an die Zeichengröße von acht mal acht Pixeln gebunden.

Die Maschinencodetabellen mit den Fensterparametern und den Zwischenspeichern fangen bei Adresse $B004 (dezimal 45060) an. Die Tabellenadressen des ersten Fensters befinden sich in $B000 und $B001 (dezimal 45056 und 45057). Da die Tabelle für jedes Fenster 11 Bytes benötigt, fängt die nächste Tabelle bei $B00F (dezimal 45071) an und die des dritten Fensters bei $B01A (dezimal 45082).

Das Beispielprogramm in BASIC steuert nur ein Fenster. Seine Daten werden von Zeile 180 bis 230 mit POKE in den Speicher geladen und in Zeile 240 initialisiert. Jedes weitere Fenster kann auf die gleiche Weise definiert werden. Wenn Sie die Adresse der gewünschten Fenstertabelle in Speicherstelle WT und WT+1 POKEn, können Sie von einem Fenster zum anderen springen. Die Richtung des Scrollens läßt sich mit einem POKE-Befehl auf die Speicherstelle WNDWTB + DIR steuern: POKE 0 scrollt links, 1 rechts, 2 auf und 3 ab.

Das Assemblerprogramm definiert zunächst eine Reihe von Konstanten. PIXADR ist eine im ROM gespeicherte Unterroutine des Spectrum. Sie berechnet die Adresse des Bildschirmbytes und innerhalb dieses Bytes die Bitnummer des Bildschirmpunktes, der von den PLOT-Koordinaten definiert wurde. PIXADR übernimmt die Y-Koordinate aus Register B und die X-Koordinate aus C und speichert die errechnete Bildschirmadresse nach dem Ablauf in Register HL und die Bitposition in A.

Die Routine INITW überprüft zunächst, ob die Koordinaten der unteren rechten Fensterecke wirklich unterhalb und rechts von der oberen linken Fensterecke liegen. Sie stellt außerdem sicher, daß der linke und rechte Rand sich nicht im gleichen Byte des Bildschirmspeichers befinden. Damit wird sichergestellt, daß das Fenster mindestens ein Zeichen breit ist.

Der letzte Abschnitt von INITW berechnet LFTMSK und RTMASK. Diese beiden Werte werden benötigt, wenn beim Scrollen des Fensterinhalts die Bildschirmbytes teilweise innerhalb und teilweise außerhalb des Fensters liegen. Die Maskenbits, die den außerhalb des Fensters liegenden Bildschirmbits entsprechen, stehen auf Eins, die zum Fensterinneren gehörenden Bits dagegen auf Null.

Da der Ablauf für eine Links- und Rechtsbewegung fast identisch ist, gibt es dafür nur eine Routine. Dabei wird über Bit Null des Richtungsbytes festgestellt, welcher Teil des Codes ausgeführt werden soll.

Da das rechte und auch das linke Scrollen bei der obersten Pixelzeile des Fensters anfängt, kopiert HORIZ zuerst die oberste Zeile in den Zwischenspeicher. Das Linksscrollen fängt dann am rechten Rand jeder Pixelzeile des Fensters an und setzt sich nach links fort. Zur Vorbereitung dieses Ablaufs werden RMASK und LMASK auf MASK1 und MASK2 kopiert, und weiterhin wird die Adresse des Bildschirmbytes am linken Rand der bearbeiteten Pixelzeile berechnet und im Registerpaar DE gespeichert. Die Adresse des Bildschirmbytes am rechten Ende der Pixelzeile wird ebenfalls berechnet und in HL abgelegt. Danach führt HLNSCR das Scrollen dieser Pixelzeile aus. Die Routine testet dabei zunächst, ob die unterste Zeile des Fensters erreicht ist. Ist dies nicht der Fall, nimmt sie die nächste Pixelzeile und springt zum Weiterscrollen auf HORIZ3.

Scroll-Vorgänge

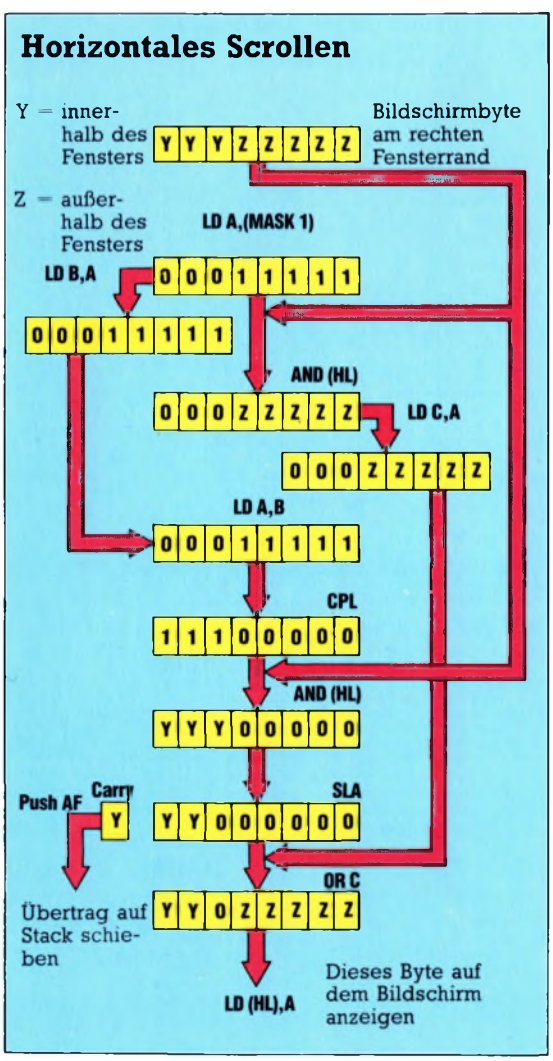
HLNSCR beginnt bei dem Randbyte, dessen Bits sowohl in als auch außerhalb des Fensters liegen können, nimmt dann die innerhalb des Fensters liegenden Bytes und hört am anderen Fensterrand auf. Unser Schaubild zeigt, wie der Code beim Linksscrollen das rechtsliegende Byte behandelt. Der Teil von HLNSCR, der bei NEXT anfängt, bewegt die Bytes innerhalb des Fensters. Das Bit, das beim Bewegen des vorigen Bytes in den Übertrag gewandert war, wird dabei mit PUSH AF auf den Stack geschoben und danach mit POPAF wieder in das Übertragsflag zurückgesetzt. Das Linksscrollen wird nun mit dem Befehl RL(HL) ausgeführt, wobei das Bildschirmbyte mit SHIFT nach links bewegt wird, das Bit des Übertragsflags linksaußen in das Byte eintritt und das rechtsaußen liegende Bit in das Übertragsflag wandert. PUSH AF speichert das Übertragsflag, so daß dieses Bit für das nächste Byte er-

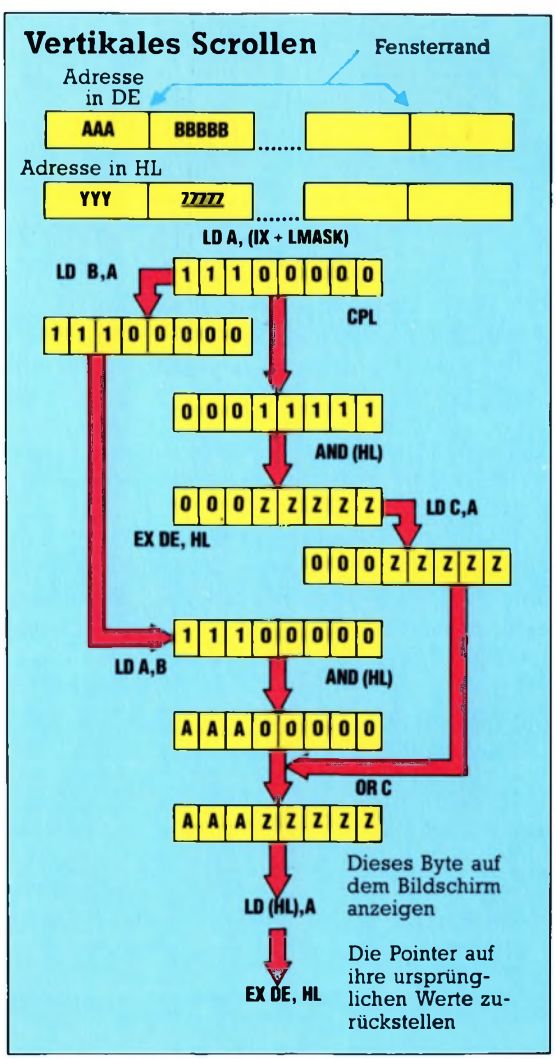
halten bleibt. Als Test für das Zeilenende braucht die Routine nur die Register L und E zu vergleichen, da das höherwertige Byte einer Bildschirmadresse bei allen Bildschirmbytes der gleichen Zeile übereinstimmt. Das Scrollen des letzten Randbytes wird wie beim ersten Randbyte durchgeführt.

Auch die Vorgänge bei der Auf- und Abwartsbewegung lassen sich mit einer Routine erledigen. Beim Aufwärtsscrollen mit VERT wird zunächst die Y-Koordinate der obersten Zeile in den Zwischenspeicher geladen, dann die Bildschirmadresse für den linken und rechten Rand der Zeile berechnet und die Zeilenlänge bestimmt. Die Routine speichert die Adresse des rechten Randes in DE und die Adresse des entsprechenden Bytes der darunterliegenden Zeile in HL. Nachdem die Unterroutine VLNSCR das Scrollen ausgeführt hat, testet VERT, ob der untere Rand des Fensters erreicht ist. Ist dies nicht der Fall, wird die darunterliegende Zeile angesprochen und VERT5 für das Scrollen dieser Zeile aufgerufen. Wenn der untere Rand erreicht ist, füllt der bei CLREDG beginnende Code die unterste Pixelzeile mit Nullen, um diese Zeile auf dem Bildschirm zu löschen.

Ebenso wie HLNSCR behandelt VLNSCR die Ränder mit einer separaten Routine. Unser zweites Bild zeigt die Bearbeitung der Randbytes. Für das Scrollen des Mittelteils der Pixelzeile werden die Register HL und DE inkrementiert, so daß sie für das erste innere Byte der laufenden Zeile auf das entsprechende Byte der darüberliegenden Zeile zeigen. VLNSCR berechnet dann die Länge des Mittelteils (d. h. der Bytes, die vollständig innerhalb des Fensters liegen), lädt diese Information in BC und bewegt den gesamten Mittelteil dieser Zeile mit dem Befehl LDIR als Block um eine Zeile nach oben.

Die Scrollroutinen sind nicht besonders schnell. Die Ursache dafür liegt in der Kombination der Links-Rechts- und Auf-Ab-Routinen und an den zahlreichen Tests, mit denen das Programm erst herausfinden muß, welchen Teil des Codes es ausführen soll. Zusätzlich verlangsamen die geteilten linken und rechten Randbytes mit ihren Spezialroutinen den Ablauf. Das Scrollen ließe sich weitaus schneller ausführen, wenn jede Bewegungsrichtung eine eigene Routine hätte und die Ränder der Fenster nur auf der Grenze zwischen zwei Bildschirmbytes liegen dürften.





Beim horizontalen Scrollen müssen die Bildschirmbytes am Fensterrand in „Masken" gesetzt werden, damit die Pixel innerhalb des Fensters von denen außerhalb trennbar sind. Der Inhalt dieser Bytes darf außerdem nur bitweise bewegt werden. Beide Vorgänge setzen für diese Aufgabe das logische AND, OR und SHIFT ein und erhalten den Inhalt des PSR durch eine Zwischenspeicherung im Stack. Das vertikale Scrollen wird durch die Memory Map des Spectrum sehr vereinfacht. Auch hier müssen die Bildschirmbytes „separiert" werden, um zwischen den Pixeln innerhalb und außerhalb des Fensters unterscheiden zu können. Der Inhalt der Register DE und HL, die die Pointer der Bildschirmadressen enthalten, werden mit dem Befehl EX ausgetauscht.

Fenster auf dem Spectum

So sollten Sie vorgehen:
1) Geben Sie das BASIC-Beispiel ein
2) SAVE „SCROLL" LINE 5
3) Tippen Sie den Maschinencode ein, und rufen Sie das Ladeprogramm für Maschinencode mit RUN auf
4) SAVE „SCROLLMC" CODE 45312,410 speichert den Code auf der Cassette direkt hinter dem BASIC-Programm
5) Spulen Sie die Cassette an den Anfang zurück, und geben sie LOAD „SCROLL" ein

Assemblerlisting

```
 10 PIXADR EQU  #22AA
 20 WT     EQU  #B000
 30 MASK1  EQU  #B002
 40 MASK2  EQU  #B003
 50 WNDWTB EQU  #B004
 60 LEFTX  EQU  0
 70 TOPY   EQU  1
 80 RIGHTX EQU  2
 90 BOTY   EQU  3
100 LBIT   EQU  4
110 RBIT   EQU  5
120 CURNTY EQU  6
130 DIR    EQU  7
140 LMASK  EQU  8
150 RMASK  EQU  9
160 LENGTH EQU  10
170        ORG  #B100
180 INIT   LD   HL,(WT)
190        PUSH HL
200        POP  IX
210        CALL INITW
220        RET
230 SCROLL LD   HL,(WT)
240        PUSH HL
250        POP  IX
260        BIT  1,(IX+DIR)
270        PUSH AF
280        CALL Z,HORIZ
290        POP  AF
300        CALL NZ,VERT
310        RET
320 HORIZ  LD   A,(IX+TOPY)
330        LD   (IX+CURNTY),A
340        BIT  0,(IX+DIR)
350        JR   Z,HORIZ1
360        LD   B,(IX+LMASK)
370        LD   A,(IX+RMASK)
380        JR   HORIZ2
390 HORIZ1 LD   B,(IX+RMASK)
400        LD   A,(IX+LMASK)
410 HORIZ2 LD   (MASK2),A
420        LD   A,B
430        LD   (MASK1),A
440 HORIZ3 LD   C,(IX+LEFTX)
450        LD   B,(IX+CURNTY)
460        CALL PIXADR
470        EX   DE,HL
480        LD   C,(IX+RIGHTX)
490        LD   B,(IX+CURNTY)
500        CALL PIXADR
510        BIT  0,(IX+DIR)
520        JR   Z,HORIZ4
530        EX   DE,HL
540 HORIZ4 CALL HLNSCR
550        LD   A,(IX+BOTY)
560        CP   (IX+CURNTY)
570        RET  Z
580        DEC  (IX+CURNTY)
590        JR   HORIZ3
600 HLNSCR LD   A,(MASK1)
610        LD   B,A
620        AND  (HL)
630        LD   C,A
640        LD   A,B
650        CPL
660        AND  (HL)
670        BIT  0,(IX+DIR)
680        JR   Z,HLN1
690        SRA  A
700        JR   HLN2
710 HLN1   SLA  A
720 HLN2   PUSH AF
730        OR   C
740        LD   (HL),A
750 NEXT   BIT  0,(IX+DIR)
760        JR   Z,HLN3
770        INC  HL
780        JR   HLN4
790 HLN3   DEC  HL
800 HLN4   LD   A,L
810        CP   E
820        JR   Z,LAST
830        POP  AF
840        BIT  0,(IX+DIR)
850        JR   Z,HLN5
860        RR   (HL)
870        JR   HLN6
880 HLN5   RL   (HL)
890 HLN6   PUSH AF
900        JR   NEXT
910 LAST   LD   C,(HL)
920        LD   A,(MASK2)
930        AND  C
940        LD   B,A
950        POP  AF
960        BIT  0,(IX+DIR)
970        JR   Z,HLN7
980        RR   C
 990        JR   HLNB
1000 HLN7  RL   C
1010 HLNB  LD   A,(MASK2)
1020       CPL
1030       AND  C
1040       OR   B
1050       LD   (HL),A
1060       RET
1070 VERT  LD   C,(IX+LEFTX)
1080       BIT  0,(IX+DIR)
1090       JR   Z,VERT1
1100       LD   B,(IX+BOTY)
1110       JR   VERT2
1120 VERT1 LD   B,(IX+TOPY)
1130 VERT2 LD   (IX+CURNTY),B
1140       CALL PIXADR
1150       PUSH HL
1160       PUSH HL
1170       LD   C,(IX+RIGHTX)
1180       LD   B,(IX+CURNTY)
1190       CALL PIXADR
1200       POP  DE
1210       AND  A
1220       SBC  HL,DE
1230       LD   A,L
1240       DEC  A
1250       LD   (IX+LENGTH),A
1260 VERT3 BIT  0,(IX+DIR)
1270       JR   Z,VERT4
1280       INC  (IX+CURNTY)
1290       JR   VERT5
1300 VERT4 DEC  (IX+CURNTY)
1310 VERT5 LD   C,(IX+LEFTX)
1320       LD   B,(IX+CURNTY)
1330       CALL PIXADR
1340       POP  DE
1350       PUSH HL
1360       CALL VLNSCR
1370       LD   A,(IX+CURNTY)
1380       BIT  0,(IX+DIR)
1390       JR   Z,VERT6
1400       CP   (IX+TOPY)
1410       JR   VERT7
1420 VERT6 CP   (IX+BOTY)
1430 VERT7 JR   NZ,VERT3
1440 CLREDG POP  HL
1450       LD   A,(IX+LMASK)
1460       AND  (HL)
1470       LD   (HL),A
1480       LD   B,(IX+LENGTH)
1490       LD   A,0
1500 CLR1  INC  HL
1510       LD   (HL),A
1520       DJNZ CLR1
1530       INC  HL
1540       LD   A,(IX+RMASK)
1550       AND  (HL)
1560       LD   (HL),A
1570       RET
1580 VLNSCR LD  A,(IX+LMASK)
1590       CALL ENDBYT
1600       INC  HL
1610       INC  DE
1620       LD   B,0
1630       LD   C,(IX+LENGTH)
1640       LDIR
1650       LD   A,(IX+RMASK)
1660 ENDBYT LD  B,A
1670       CPL
1680       AND  (HL)
1690       LD   C,A
1700       EX   DE,HL
1710       LD   A,B
1720       AND  (HL)
1730       OR   C
1740       LD   (HL),A
1750       EX   DE,HL
1760       RET
1770 INITW LD   A,(IX+RIGHTX)
1780       CP   (IX+LEFTX)
1790       JR   Z,ERROR
1800       JR   C,ERROR
1810       LD   A,(IX+TOPY)
1820       CP   (IX+BOTY)
1830       JR   Z,ERROR
1840       JR   C,ERROR
1850       LD   C,(IX+LEFTX)
1860       LD   B,(IX+TOPY)
1870       CALL PIXADR
1880       PUSH HL
1890       LD   (IX+LBIT),A
1900       LD   C,(IX+RIGHTX)
1910       LD   B,(IX+TOPY)
1920       CALL PIXADR
1930       LD   (IX+RBIT),A
1940       POP  BC
1950       LD   A,C
1960       CP   L
1970       JR   Z,ERROR
1980 LFTMSI LD  B,(IX+LBIT)
1990       LD   A,0
2000 L1    SCF
2010       RRA
2020       DJNZ L1
2030       LD   (IX+LMASK),A
2040 RTMASK LD  B,(IX+RBIT)
2050       LD   A,255
2060 L2    AND  A
2070       RRA
2080       DJNZ L2
2090       LD   (IX+RMASK),A
2100       RET
2110 ERROR RST  8
2120       DEFB 25
```

BASIC-Beispiel

```
  5 CLEAR 32767
 10 LOAD ""CODE
 20 LET WT=45056
 30 LET WINDOWTABLE=45060: REM B004 HEX
 40 LET LEFTX=0
 50 LET TOPY=1
 60 LET RIGHTX=2
 70 LET BOTY=3
 75 LET DIR=7
 80 LET SCROLL=45322
 90 LET INIT=45312
100 BORDER 6
110 PAPER 4: INK 2
120 CLS
180 POKE WT,4: POKE WT+1,176
190 REM WT & WT+1 NOW CONTAIN ADDRESS 45060
IN LO,HI FORMAT
200 POKE WINDOWTABLE+LEFTX,5
210 POKE WINDOWTABLE+TOPY,80
220 POKE WINDOWTABLE+RIGHTX,250
230 POKE WINDOWTABLE+BOTY,35
240 RANDOMIZE USR INIT
250 FOR Y=0 TO 175
260 PLOT 0,Y
270 DRAW 255,0
280 NEXT Y
290 POKE WINDOWTABLE+DIR,2
300 FOR Y=0 TO 45
310 RANDOMIZE USR SCROLL
320 NEXT Y
400 PRINT AT 12,20;"HELLO";
410 POKE WINDOWTABLE+DIR,0
420 FOR I=1 TO 130
430 RANDOMIZE USR SCROLL
440 NEXT I
470 POKE WINDOWTABLE+DIR,3
480 FOR I=1 TO 35
490 RANDOMIZE USR SCROLL
500 NEXT I
510 POKE WINDOWTABLE+DIR,1
520 FOR I=1 TO 130
530 RANDOMIZE USR SCROLL
540 NEXT I
550 POKE WINDOWTABLE+DIR,2
560 FOR I=1 TO 35
570 RANDOMIZE USR SCROLL
580 NEXT I
590 GO TO 410
999 STOP
```

Ladeprogramm für Maschinencode

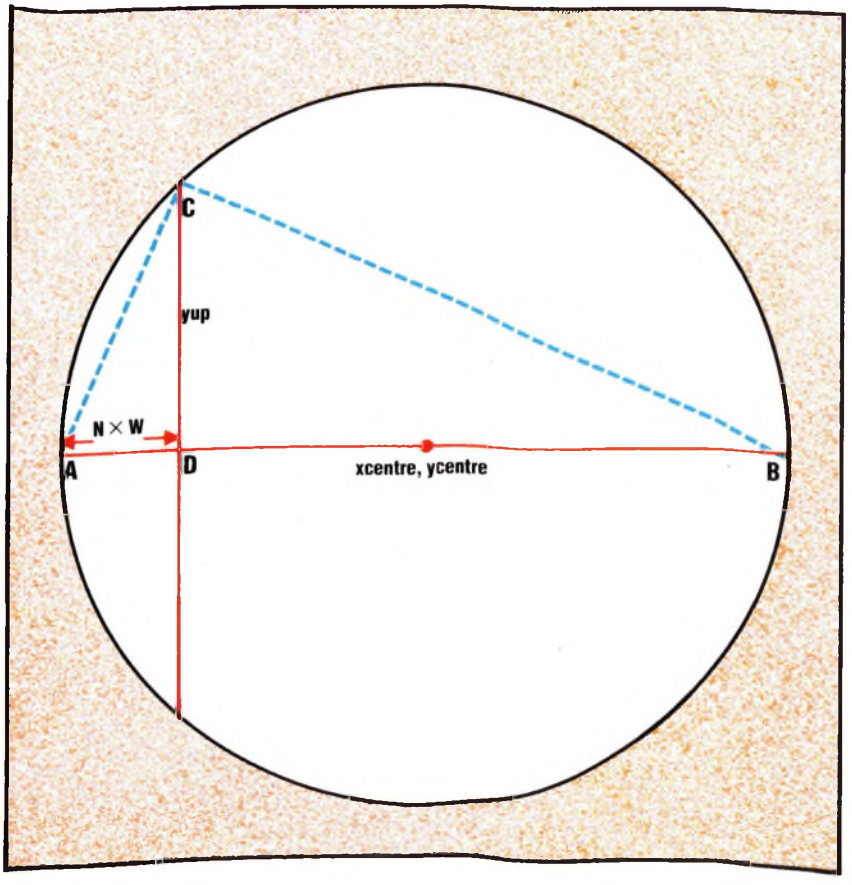
```
 100 LET a=45312
 110 FOR i=1000 TO 1500 STEP 10
 120 LET s=0
 125 FOR a=a TO a + 7
 130 READ b
 140 POKE a,b
 150 LET s=s+b
 160 NEXT a
 170 READ c
 180 IF s<>c THEN PRINT "ERROR IN LINE "; i : STOP
 190 NEXT i
1000 DATA 42,0,176,229,221,225,205,69,1167
1010 DATA 178,201,42,0,176,229,221,225,1272
1020 DATA 221,203,7,78,245,204,29,177,1164
1030 DATA 241,196,184,177,201,221,126,1,1347
1040 DATA 221,119,6,221,203,7,70,40,887
1050 DATA 8,221,70,8,221,126,9,24,687
1060 DATA 6,221,70,9,221,126,8,50,711
1070 DATA 3,176,120,50,2,176,221,78,826
1080 DATA 0,221,70,6,205,170,34,235,941
1090 DATA 221,78,2,221,70,6,205,170,973
1100 DATA 34,221,203,7,70,40,1,235,811
1110 DATA 205,103,177,221,126,3,221,190,1246
1120 DATA 6,200,221,53,6,24,215,58,783
1130 DATA 2,176,71,166,79,120,47,166,827
1140 DATA 221,203,7,70,40,4,203,47,795
1150 DATA 24,2,203,39,245,177,119,221,1030
1160 DATA 203,7,70,40,3,35,24,1,383
1170 DATA 43,125,187,40,16,241,221,203,1076
1180 DATA 7,70,40,4,203,30,24,2,380
1190 DATA 203,22,245,24,226,78,58,3,859
1200 DATA 176,161,71,241,221,203,7,70,1150
1210 DATA 40,4,203,25,24,2,203,17,518
1220 DATA 58,3,176,47,161,176,119,201,941
1230 DATA 221,78,0,221,203,7,70,40,840
1240 DATA 5,221,70,3,24,3,221,70,617
1250 DATA 1,221,112,6,205,170,34,229,978
1260 DATA 229,221,78,2,221,70,6,205,1032
1270 DATA 170,34,209,167,237,82,125,61,1085
1280 DATA 221,119,10,221,203,7,70,40,891
1290 DATA 5,221,52,6,24,3,221,53,585
1300 DATA 6,221,78,0,221,70,6,205,807
1310 DATA 170,34,209,229,205,40,178,221,1286
1320 DATA 126,6,221,203,7,70,40,5,678
1330 DATA 221,190,1,24,3,221,190,3,853
1340 DATA 32,209,225,221,126,8,166,119,1106
1350 DATA 221,70,10,62,0,35,119,16,533
1360 DATA 252,35,221,126,9,166,119,201,1129
1370 DATA 221,126,8,205,58,178,35,19,850
1380 DATA 6,0,221,78,10,237,176,221,949
1390 DATA 126,9,71,47,166,79,235,120,853
1400 DATA 166,177,119,235,201,221,126,2,1247
1410 DATA 221,190,0,40,67,56,65,221,860
1420 DATA 126,1,221,190,3,40,57,56,694
1430 DATA 55,221,78,0,221,70,1,205,851
1440 DATA 170,34,229,221,119,4,221,78,1076
1450 DATA 2,221,70,1,205,170,34,221,920
1460 DATA 119,5,193,121,189,40,25,221,913
1470 DATA 70,4,62,0,55,31,16,252,490
1480 DATA 221,119,8,221,70,5,62,255,961
1490 DATA 167,31,16,252,221,119,9,201,1016
1500 DATA 207,25,0,0,0,0,0,0,232
```

Computerkreise

Mit den Maschinenroutinen Plotsub und Linesub haben wir die hochauflösenden Fähigkeiten des Commodore 64 für unsere Zwecke eingesetzt. In diesem Artikel wird eine Routine gezeigt, die bei Angabe der Mittelpunktkoordinaten und des Radius Kreise zeichnet.

Heimcomputer können keine exakten Kreise erzeugen. Wie nahe die Zeichnung einem perfekten Kreis kommt, bestimmen hauptsächlich die dafür eingesetzte Methode sowie die Länge und Komplexität der Berechnungsroutine. Unter BASIC werden die Koordinaten der Kreispunkte normalerweise mit Sinus- und Cosinusfunktionen und Quadratwurzeln berechnet. Da sich diese beiden Methoden jedoch nur schwer in Maschinencode umsetzen lassen, stellen wir hier eine Methode vor, die speziell für diesen Code geeignet ist.

Dabei wird der Kreisdurchmesser in eine Anzahl gleicher Abschnitte mit der Länge W unterteilt. Von jedem Teilungspunkt reicht eine Linie hinauf bis zu Punkt C der Kreislinie. Die Abbildung zeigt eine dieser Linien, die N Unterteilungen vom linken Ende des Durchmessers AB entfernt ist. Bei der Verbindung der Punkte A und B mit C entstehen die beiden rechtwinkligen Dreiecke ACD und BCD.



Aus der Darstellung lassen sich nach dem Satz des Pythagoras folgende Formeln ableiten:

$AC^2 = AD^2 + CD^2$

$CB^2 = DB^2 + CD^2$

Die Addition der beiden Gleichungen ergibt:

$AC^2 + CB^2 = AD^2 + DB^2 + 2CD^2$

Da bei Kreisen das Dreieck ABC jedoch ebenfalls rechtwinklig ist, können wir sagen:

$AC^2 + CB^2 = AB^2$

Die modifizierte Gleichung setzt sich wie folgt zusammen:

$AB^2 = AD^2 + DB^2 + 2CD^2$

CD entspricht „yup" – dem Abstand des Durchmessers zu der Kreislinie. AD ist N×W und AB ist 2×R (R = Kreisradius). Nachdem wir die Werte in die Gleichung eingesetzt und diese umgeformt haben, erhalten wir:

$2yup^2 = (2R)^2 - (NW)^2 - (2R-NW)^2$

$yup^2 = 2RNW - N^2 W^2$

Wenn wir nun den Durchmesser in 64 gleiche Abschnitte unterteilen, dann ist $W = 2*R/64$, was sich auch als $W = R/32$ schreiben läßt. In die Gleichung eingesetzt ergibt sich:

$yup^2 = 2RNR/32 - N^2 R^2/32^2$

$yup^2 = R^2/32^2 \times (64N-N^2)$

Nachdem auf beiden Seiten die Quadratwurzel gezogen wurde, haben wir:

$yup = R/32*SQR(64N-N^2)$

Wenn wir mit x = xcentre – R anfangen und x in 64 gleichen Schritten inkrementieren, dann liefert uns die Gleichung bei jedem Schritt die Länge der Strecke vom Durchmesser bis zur Kreislinie, wobei N die Anzahl der Schritte angibt. Beachten Sie, daß dieses Ergebnis unabhängig von den Koordinaten des Kreismittelpunktes oder dem Radius ist. Wir können daher eine Tabelle zusammenstellen, die für jeden N-Wert von 0 bis 64 die Lösung der Wurzelfunktion enthält. Da diese Tabelle nur einmal erstellt werden muß, können wir sie fest in das Programm einbauen.

Die absolute Y-Koordinate für jedes Inkrement von X ist:

ya = ycentre – yup

Aufgrund der Symmetrie des Kreises können wir die Y-Koordinaten für die untere Kreishälfte folgendermaßen berechnen:

yb = ycentre + yup

Das Flußdiagramm zeigt den Ablauf bei der Berechnung der Kreispunkte. Da für jeden Punkt nur eine Multiplikation ausgeführt wird,

ist die Routine relativ schnell. Dabei ergeben sich jedoch zwei Probleme. Zunächst entsteht kein kontinuierlicher Kreis, sondern nur eine Anzahl Punkte auf der Kreislinie. Weiterhin erzeugt diese Methode im Maschinencode einige Ungenauigkeiten, die unter BASIC nicht auftreten.

Das erste Problem läßt sich lösen, indem man die Punkte über Linesub mit kurzen Linien verbindet und so eine fortlaufende Kreislinie erzeugt. Das zweite Problem entsteht durch Ungenauigkeiten der Quadratwurzeltabelle. Wenn wir die in BASIC erzeugten Werte in die dafür reservierten Adreßbytes POKEn, werden nur die Ganzzahlen gespeichert. Wir müssen daher die Genauigkeit der Tabelle erhöhen, indem wir jede Zahl mit Acht multiplizieren. Da 32 die höchste Zahl der Tabelle ist, lassen sich die multiplizierten Werte immer noch in einem einzigen Byte unterbringen. Um die Routine einfach zu halten, haben wir allerdings nur Näherungswerte eingesetzt. Die Maschinencoderoutine teilt den Tabellenwert mit drei LSR-Befehlen (logical shifts right) durch Acht. Wichtig ist dabei, daß wir den Teilungsrest für unsere weiteren Berechnungen zur Verfügung haben.

Das Circsub-Programm

Der Quellentext reserviert am Anfang 65 Bytes für die Tabelle. Dabei wird die erste Byte-Adresse als Label angegeben, und alle folgenden Tabelleneinträge werden über die indizierte Adressierung angesprochen. Bei der Eingabe und Assemblierung des Quellentextes ist nichts Besonderes zu beachten. Bevor der Code gespeichert wird, sollte jedoch das folgende BASIC-Programm eingegeben und mit RUN aufgerufen werden. Der Maschinencode wird dadurch nicht beeinflußt, da er sich oben im Speicher befindet. Das BASIC-Programm erzeugt die 65 Tabellenwerte für unser Circsub-Programm, multipliziert die Werte mit Acht und POKEt die Ergebnisse in den dafür vorgesehenen Speicherbereich. Nach Generierung der Tabelle kann der Maschinencode mit SAVE gespeichert werden. Vergewissern Sie sich aber, daß dabei die Tabelle in den Objektcode mit eingeschlossen wird (die Tabelle beginnt bei der Adresse $C500). Bei jedem Aufruf von Circsub wird die Tabelle dann automatisch geladen.

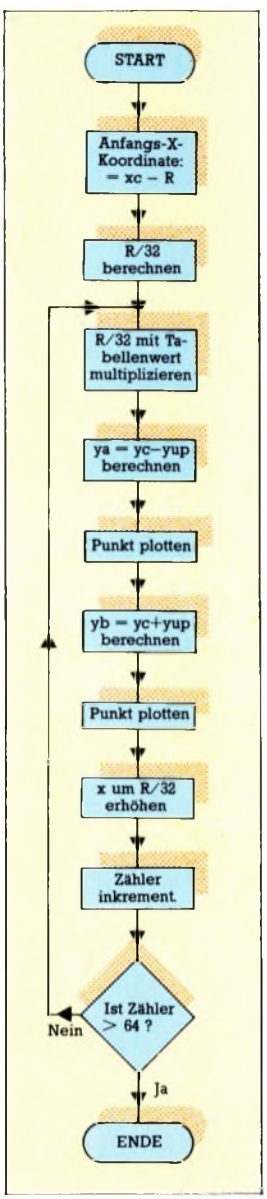
```
5 REM♦♦♦♦ ERZEUGE CIRCSUB TABLE ♦♦♦♦
10 FORN=0TO64
20 X=SQR(.64*N-INT(.2*N
25 X%=X: IF=X-X%
27 IFRE(.5THENX%=X%+1
28 IFX%>255THENX%=255
29 POKE50432+N,X%
30 NEXT
```

Das folgende Programm zeigt, wie Circsub von einem BASIC-Programm aus aufgerufen wird. Dabei müssen die Koordinaten des Kreismittelpunktes und der Radius angegeben werden. Die Unterroutine in Zeile 2000 wandelt die X-Koordinate in das Format LO-Byte/HI-Byte um, übergibt diese Werte mit POKE an Circsub und ruft die Routine mit SYS auf. Beachten Sie, daß Circsub die Linesub-Routine einsetzt und diese wiederum Plotsub. Alle drei Unterroutinen müssen daher am Anfang des Programms geladen werden. Das BASIC-Programm zeichnet Kreise mit sich ständig vergrößernden Radien auf den Bildschirm.

```
3 REM♦♦♦♦ CIRCSUB TEST PROGRAMM ♦♦♦♦
5 DN=8 REM FUR CASSETTE DN=1
10 IFA=0THENA=1:LOAD"PLOTSUB.HEX",DN,1
15 IFA=1THENA=2:LOAD"LINESUB.HEX",DN,1
20 IFA=2THENA=3:LOAD"CIRCSUB.HEX",DN,1
100 GOSUB1000
110 YC=100:R=2
120 FORXC=30TO210STEP7
130 R=R+3
140 GOSUB2000
150 NEXT
160 GETA$:IFA$=""THEN160
170 GOSUB3000
180 END
1000 REM ♦♦♦♦ SET HIRES ♦♦♦♦
1010 POKE49408,1:POKE49409,1
1020 POKE49410,3
1030 SYS49422
1040 RETURN
1050 .
2000 REM ♦♦♦♦ ENTER CIRCSUB ♦♦♦♦
2010 CHI=INT(XC/256):CLO=XC-256*CHI
2020 POKE50497,CLO:POKE50498,CHI
2030 POKE50499,YC
2040 POKE50500,R
2050 SYS50521
2060 RETURN
2070 .
3000 REM ♦♦♦♦ CLEAR HIRES ♦♦♦♦
3005 RESTORE
3007 PRINTCHR$(147) REM CLEAR SCREEN
3008 PRINTTAB(10)"CIRCSUB VARIABLES"
3009 PRINT
3010 POKE49408,0:SYS49422
3030 FORI=50497TO50497+23STEP2
3035 READA$
3040 PRINTTAB(2)A$,PEEK(I)
3045 READA$
3047 PRINTA$,PEEK(I+1)
3050 NEXT
3060 RETURN
3070 .
5000 DATAXCLO,XCHI,YC,R,RADIUS,INTR
5010 DATAREMR,TOTX,RESPEM,RESLO,RESHI,OLDXLO
5020 DATAOLDXHI,NEWXLO,NEWXHI,OLDYA
5030 DATAOLDYB,NEWYA,NEWYB,XFLAG,YFLAG,OLDYA,NEWY
5040 DATAINTTMP,KEMTMP
```

Wenn Sie keinen Assembler haben, können Sie den Maschinencode – wie auch die anderen hochauflösenden Grafikroutinen für den Commodore 64 – in Form von Data-Befehlen eingeben. In diesem Fall sollte der Maschinencode mit dem abgebildeten BASIC-Ladeprogramm in den Arbeitsspeicher geladen werden, sowie Linesub und Plotsub mit ihren jeweiligen Programmen. Die Ladeprogramme müssen vor dem Aufruf des BASIC-Beispielprogramms einmal abgelaufen sein. Da alle drei Routinen sich nun bereits im Speicher befinden, können die Zeilen 10, 15 und 20 des Beispielprogramms gelöscht werden. Beachten Sie weiterhin, daß das BASIC-Ladeprogramm für Circsub die Quadratwurzeltabelle bereits enthält und das Generierungsprogramm somit nicht aufgerufen werden muß.

In den nächsten Folgen werden wir uns mit dem 6809-Prozessor beschäftigen, der auch bei den Dragon-Computern verwendet wird.



BASIC-Ladeprogramm

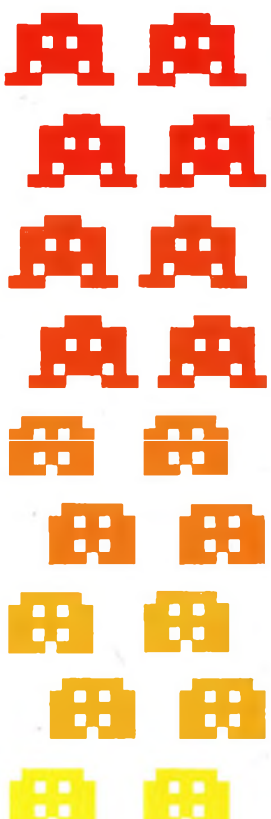
```
10 REM**** BASIC LOADER FOR CIRCSUB ****
20 FORI=50432TO50432+498
30 READA:POKEI,A
40 CC=CC+A
50 NEXT I
60 READA:IFA<>CCTHENPRINT"CHECKSUM ERROR"
100 DATA0,63,89,108,123,137,149,159
110 DATA169,177,185,193,199,205,211
120 DATA216,221,226,230,233,237,240
130 DATA243,245,247,249,251,252,253
140 DATA254,255,255,255,255,255,254
150 DATA253,252,251,249,247,245,243
160 DATA240,237,233,230,226,221,216
170 DATA211,205,199,193,185,177,169
180 DATA159,149,137,123,108,89,63,0
190 DATA205,0,100,80,2,16,16,0,0,0,29
200 DATA1,31,1,100,100,100,100,0,0,120
210 DATA100,0,0,72,138,72,152,72,173
220 DATA68,197,41,31,141,70,197,173,68
230 DATA197,160,5,74,136,208,252,141
240 DATA69,197,173,65,197,56,237,68
250 DATA197,141,77,197,141,75,197,173
260 DATA66,197,233,0,141,78,197,141,76
270 DATA197,173,67,197,141,79,197,141
280 DATA80,197,169,0,170,141,71,197
290 DATA189,0,197,160,3,74,136,208,252
300 DATA141,87,197,189,0,197,41,7,141
310 DATA88,197,172,68,197,169,0,141,72
320 DATA197,141,73,197,141,74,197,173
330 DATA72,197,24,109,88,197,141,72
340 DATA197,201,8,144,23,56,233,8,141
350 DATA72,197,173,73,197,24,105,1,141
360 DATA73,197,173,74,197 105,0,141,74
370 DATA197,173,73,197,24,109,87,197
380 DATA141,73,197,173,74,197,105,0
390 DATA141,74,197,136,208,198,160,5
400 DATA78,74,197,110,73,197,110,72
410 DATA197,136,208,244,173,72,197,201
420 DATA16,144,9,173,73,197,24,105,1
430 DATA141,73,197,173,67,197,56,237
440 DATA73,197,141,81,197,173,81,197
450 DATA141,86,197,173,79,197,141,85
460 DATA197,32,165,198,173,67,197,24
470 DATA109,73,197,141,82,197,173,82
480 DATA197,141,86,197,173,80,197,141
490 DATA85,197,32,165,198,173,77,197
500 DATA141,75,197,173,78,197,141,76
510 DATA197,173,81,197,141,79,197,173
520 DATA82,197,141,80,197,173,71,197
530 DATA24,109,70,197,141,71,197,201
540 DATA32,144,26,173,71,197,56,233,32
550 DATA141,71,197,173,77,197,24,105,1
560 DATA141,77,197,173,76,197,105,0
570 DATA141,78,197,173,77,197,24,109
580 DATA69,197,141,77,197,173,78,197
590 DATA105,0,141,78,197,232,224,65
600 DATA240,3,76,153,197,104,168,104
610 DATA170,104,96,169,0,141,83,197
620 DATA141,84,197,173,75,197,205,77
630 DATA197,208,3,238,83,197,173,85
640 DATA197,205,86,197,208,3,238,84
650 DATA197,173,83,197,45,84,197,208
660 DATA39,173,75,197,141,0,195,173,76
670 DATA197,141,1,195,173,85,197,141,4
680 DATA195,173,77,197,141,2,195,173
690 DATA78,197,141,3,195,173,86,197
700 DATA141,5,195,32,14,195,96
710 DATA67223 REM*CHECKSUM*
```

Circsub-Programm

```
;******************
;******************
;**              **
;**  CIRCSUB 64  **
;**              **
;******************
;******************
;**** LINESUB VARIABLES ****
;
LINSUB = $C38E
X1LO   = $C300
X1HI   = $C301
X2LO   = $C302
X2HI   = $C303
Y1     = $C304
Y2     = $C305
*      = $C500
;
;**** CIRCSUB VARIABLES ****
;
TABLE  *=*+65          ; LOOK UP TABLE
XCTRLO *=*+1
XCTRHI *=*+1
YCTR   *=*+1
RADIUS *=*+1
INTR   *=*+1
REMR   *=*+1
TOTX   *=*+1
RESREM *=*+1
RESLO  *=*+1
RESHI  *=*+1
OLDXLO *=*+1
OLDXHI *=*+1
NEWXLO *=*+1
NEWXHI *=*+1
OLDYA  *=*+1
OLDYB  *=*+1
NEWYA  *=*+1
NEWYB  *=*+1
XFLAG  *=*+1
YFLAG  *=*+1
OLDY   *=*+1
NEWY   *=*+1
INTTAB *=*+1
REMTAB *=*+1
;
;**** PUSH REGISTERS ONTO STACK ****
;
       PHA
       TXA
       PHA
       TYA
       PHA
;
;**** CALC INT(R/32) AND REMR ****
;
AD 44 C5       LDA RADIUS
29 1F          AND #$1F
8D 46 C5       STA REMR
AD 44 C5       LDA RADIUS
A0 05          LDY #$05
4A       LOOP1 LSR A
88             DEY
D0 FC          BNE LOOP1
8D 45 C5       STA INTR
;
;**** CALC INITIAL X COORDINATE****
;
AD 41 C5       LDA XCTRLO
38             SEC
ED 44 C5       SBC RADIUS
8D 4D C5       STA NEWXLO
8D 4B C5       STA OLDXLO
AD 42 C5       LDA XCTRHI
E9 00          SBC #$00
8D 4E C5       STA NEWXHI
8D 4C C5       STA OLDXHI
;
;**** SET INITIAL OLD Y VALUES ****
;
AD 43 C5       LDA YCTR
8D 4F C5       STA OLDYA
8D 50 C5       STA OLDYB
;
;**** ZERO COUNT AND TOTX ****
;
A9 00          LDA #$00
AA             TAX
8D 47 C5       STA TOTX
;
;**** CALC INTTAB AND REMTAB ****
;
       NEXTPT
BD 00 C5       LDA TABLE,X
A0 03          LDY #$03
       LOOP2
4A             LSR A
88             DEY
D0 FC          BNE LOOP2
8D 57 C5       STA INTTAB
;
BD 00 C5       LDA TABLE,X
29 07          AND #$07
8D 58 C5       STA REMTAB
;
;**** MULTIPLY TABLE BY RADIUS ****
;
AC 44 C5       LDY RADIUS
A9 00          LDA #$00
8D 48 C5       STA RESREM
8D 49 C5       STA RESLO       ;ZERO ANSWER
8D 4A C5       STA RESHI
;
AD 48 C5 AGAIN LDA RESREM
18             CLC
6D 58 C5       ADC REMTAB
8D 48 C5       STA RESREM
C9 08          CMP #$08        ;>= 8?
90 17          BCC NOCARR
38             SEC
E9 08          SBC #$08        ;RESET REM
8D 48 C5       STA RESREM
AD 49 C5       LDA RESLO
18             CLC
69 01          ADC #$01        ;INC RESULT
8D 49 C5       STA RESLO
AD 4A C5       LDA RESHI
69 00          ADC #$00
8D 4A C5       STA RESHI
       NOCARR
AD 49 C5       LDA RESLO
18             CLC
6D 57 C5       ADC INTTAB      ;ADD INTEGER PART
8D 49 C5       STA RESLO
AD 4A C5       LDA RESHI
69 00          ADC #$00
8D 4A C5       STA RESHI
88             DEY
D0 C6          BNE AGAIN
;
;**** DIVIDE RESULT BY 32 ****
;
A0 05          LDY #$05
       LOOP3
4E 4A C5       LSR RESHI
6E 49 C5       ROR RESLO
6E 48 C5       ROR RESREM
88             DEY
D0 F4          BNE LOOP3
;
AD 48 C5       LDA RESREM
C9 10          CMP #$10        ; REM >=16?
90 09          BCC NOCRRY
AD 49 C5       LDA RESLO
18             CLC
69 01          ADC #$01        ; ADD 1
8D 49 C5       STA RESLO
;
;**** CALC 1ST Y COORDINATE ****
;
       NOCRRY
AD 43 C5       LDA YCTR
38             SEC
ED 49 C5       SBC RESLO
8D 51 C5       STA NEWYA
;
; **** DRAW LINE ****
;
AD 51 C5       LDA NEWYA
8D 56 C5       STA NEWY
AD 4F C5       LDA OLDYA
8D 55 C5       STA OLDY
20 A5 C6       JSR DRAW
;
;**** CALC 2ND Y COORDINATE ****
;
AD 43 C5       LDA YCTR
18             CLC
6D 49 C5       ADC RESLO
8D 52 C5       STA NEWYB
;
;**** DRAW LINE ****
;
AD 52 C5       LDA NEWYB
8D 56 C5       STA NEWY
AD 50 C5       LDA OLDYB
8D 55 C5       STA OLDY
20 A5 C6       JSR DRAW
;
;**** SWOP OLD FOR NEW ****
;
AD 4D C5       LDA NEWXLO
8D 4B C5       STA OLDXLO
AD 4E C5       LDA NEWXHI
8D 4C C5       STA OLDXHI
AD 51 C5       LDA NEWYA
8D 4F C5       STA OLDYA
AD 52 C5       LDA NEWYB
8D 50 C5       STA OLDYB
;
;**** ALTER X COORDINATE ****
;
AD 47 C5       LDA TOTX
18             CLC
6D 46 C5       ADC REMR
8D 47 C5       STA TOTX        ;ADD REMR TO TOTAL
C9 20          CMP #$20        ;>=32?
90 1A          BCC NOINC
AD 47 C5       LDA TOTX
38             SEC
E9 20          SBC #$20        ;RESET TOTX
8D 47 C5       STA TOTX
AD 4D C5       LDA NEWXLO
18             CLC
69 01          ADC #$01          INC X COORD
8D 4D C5       STA NEWXLO
AD 4C C5       LDA OLDXHI
69 00          ADC #$00
8D 4E C5       STA NEWXHI
       NOINC
AD 4D C5       LDA NEWXLO
18             CLC
6D 45 C5       ADC INTR
8D 4D C5       STA NEWXLO
AD 4E C5       LDA NEWXHI
69 00          ADC #$00
8D 4E C5       STA NEWXHI
;
;**** INCREMENT COUNTER ****
;
E8             INX
E0 41          CPX #$41
F0 03          BEQ FINISH
4C 99 C5       JMP NEXTPT
;
;**** PULL REGISTERS OFF STACK ****
;
       FINISH
68             PLA
A8             TAY
68             PLA
AA             TAX
68             PLA
60             RTS
;
;**** CHECK FOR SAME POINT ****
;
       DRAW
A9 00          LDA #$00
8D 53 C5       STA XFLAG       ;ZERO FLAGS
8D 54 C5       STA YFLAG
;
AD 4B C5       LDA OLDXLO
CD 4D C5       CMP NEWXLO
D0 03          BNE NOXFLG
       NOXFLG
AD 4F C5       LDA OLDY
CD 56 C5       CMP NEWY
D0 03          BNE NOYFLG
EE 54 C5       INC YFLAG
       NOYFLG
AD 53 C5       LDA XFLAG
2D 54 C5       AND YFLAG
D0 27          BNE NODRAW
;
; **** DRAW LINE ****
;
AD 4B C5       LDA OLDXLO
8D 00 C3       STA X1LO
AD 4C C5       LDA OLDXHI
8D 01 C3       STA X1HI
AD 4F C5       LDA OLDY
8D 04 C3       STA Y1
AD 4D C5       LDA NEWXLO
8D 02 C3       STA X2LO
AD 4E C5       LDA NEWXHI
8D 03 C3       STA X2HI
AD 56 C5       LDA NEWY
8D 05 C3       STA Y2
;
20 0E C3       JSR LINSUB
;
       NODRAW
60             RTS
```

Immer mit der Ruhe!

Der Hauptvorteil des Maschinencodes ist seine hohe Ausführgeschwindigkeit. Da der Code für manche Aufgaben jedoch zu schnell ist, werden Zeitverzögerungen eingebaut. Wir untersuchen einige Methoden zur Steuerung des Z80 und des 6502.



In der Assemblersprache des 6502 gibt es mehrere Möglichkeiten, Verzögerungsschleifen zu programmieren. Am einfachsten ist das Laden eines Indexregisters, das dann von einer Schleife so lange dekrementiert wird, bis es Null erreicht hat:

V-Schleife	Zeitbedarf für jeden Ablauf
LDY #$07	Zwei Zyklen
DEY	Zwei Zyklen
BNE LOOP	Zwei Zyklen (bei Verzweigung auf die gleiche Seite 3 Zyklen, zu einer anderen 4 Zyklen)

Jede Maschinencodeanweisung benötigt eine bestimmte Anzahl Taktzyklen für die Ausführung. Informationen darüber sind normalerweise bei der Erläuterung der einzelnen Befehle zu finden. So braucht der Befehl DEY zwei Zyklen und LDY im unmittelbaren Adressierungsmodus ebenfalls zwei. Da jeder Zyklus eine Microsekunde dauert, können wir die „Echtzeit" berechnen, die die Verzögerungsschleife benötigt. Die Gesamtzahl der Zyklen errechnet sich folgendermaßen:

1. Der Befehl LDY #$07 braucht zwei Zyklen.
2. Das Programm durchläuft die Schleife siebenmal. Da der BNE-Befehl bei jeder Rückverzweigung drei Zyklen benötigt, dauern DEY und BNE (2+3)*7=35 Zyklen.
3. Der letzte BNE-Befehl verzweigt nicht zurück und braucht daher nur zwei Zyklen.

Die Gesamtzahl der Zyklen beträgt 2+35−1=36. Die Ausführung der Verzögerungsschleife dauert daher 36 Microsekunden.

Beim Einsatz von Maschinencodeschleifen für „Echtzeit"-Verzögerungen (Verzögerungen, die sich in Sekunden oder Microsekunden messen lassen) entstehen jedoch mehrere Probleme. Während der Ausführung eines Maschinencodeprogramms unterbricht der Prozessor seine Aktivität in regelmäßigen Abständen, um andere Teile des Systems zu „bedienen". So überprüft er die Tastatur auf Eingaben, aktualisiert die interne Uhr etc. Diese Unterbrechungen heißen „Interrupts". Auf dem 6502 treten zwei Arten von Interrupts auf: NMI (nicht maskierbare Interrupts) und IRQ (Interrupt-Anfragen). Der erste Interrupt-Typ kann nicht gestoppt werden. IRQ-Interrupts sind jedoch für das Funktionieren des Prozessors nicht unbedingt notwendig und lassen sich sperren.

IRQ-Interrupts werden „maskiert", indem ein bestimmtes Bit des Prozessor-Status-Registers mit dem Befehl SEI auf Eins gesetzt wird. Der CLI-Befehl setzt das gleiche Bit zurück und macht IRQ-Interrupts wieder möglich. Wenn wir nun die IRQ-Interrupts vor Aufruf der Verzögerungsschleife maskieren, können wir die Ablaufzeit präzisieren. Treten während der Schleifenausführung jedoch nicht maskierbare Interrupts auf, läßt sich die Ablaufzeit nicht mehr exakt definieren. Zur Maskierung der Interrupts sollte unsere Schleife auf folgende Art und Weise verändert werden:

Befehl	Funktion	Zeitbedarf
SEI	IRQ ausschalten	Zwei Zyklen
LDY #$07		36 Zyklen
DEY		
BNE LOOP	IRQ einschalten	
CLI		Zwei Zyklen

Die Maskierung der IRQs verlängert die Routine um vier Zyklen, so daß die gesamte Verzögerung nun 40 Microsekunden beträgt – vorausgesetzt, daß keine nicht maskierbaren Interputs auftreten.

Ein weiteres Problem von Verzögerungsschleifen ist ihre „Auflösung" – das heißt, das Intervall zwischen zwei Zählerwerten. In unserem Beispiel haben wir das Y-Register mit dem Wert Sieben geladen. Bei dem Wert Sechs beträgt die Verzögerungszeit jedoch 35 Microsekunden (2+2+(2+3)*6−1+2), bei Fünf 30 Microsekunden und so weiter bis zur minimalen Auflösung von fünf Microsekunden.

Das Programm läßt sich nun noch durch NOP-Befehle außerhalb der Schleife „verfeinern". Ein NOP-Befehl dauert zwei Zyklen. Dabei führt der Prozessor keine Operation aus (No OPeration). Für eine Verzögerung von 44 Microsekunden fügen wir daher vor (oder nach) der Schleife einfach zwei NOP-Befehle ein:

SEI	Zwei Zyklen
LDY #$07	Zwei Zyklen
NOP	Zwei Zyklen
NOP	Zwei Zyklen
DEY BNE LOOP	34 Zyklen
CLI	Zwei Zyklen

Diese Art der Verzögerung ist an eine Obergrenze gebunden, die vom maximalen Wert des Y abhängt. Da das Y-Register acht Bits enthält, liegt die Obergrenze bei 255 und die größtmögliche Verzögerung bei 1280 Microsekunden (2+2+(2+3)*255−1+2) oder etwa einer Tausendstelsekunde. Die Verzögerung kann zwar durch weitere NOP-Befehle im Inneren der Schleife noch etwas verlängert werden — für längere Zeiten müssen wir jedoch andere Methoden verwenden.

Lange Verzögerungen werden oft durch verschachtelte Schleifen oder durch Dekrementierung einer größeren Zahl programmiert. Bei jeder dieser Methoden sollte die Standardauflösung berechnet werden.

```
DELAY SEI
      LDX #$04    ;X—Reg = Zaehler Aussenschleife
LOOP1 LDY #$FF ;Y—Reg = Zaehler Innenschleife
LOOP2 DEY
      BNE LOOP2  ;Ende Innenschleife
      DEX
      BNE LOOP1  ;Ende Aussenschleife
      CLI
```

Die innere Schleife braucht 1276 Microsekunden (2+(2+3)*255−1). Die äußere Schleife steuert die innere und führt DEX und BNE viermal aus. Die Gesamtzeit beträgt: 2+2+(1 276+2+3)*4−1+2=5129 Microsekunden.

Die Befehle des Z80-Maschinencodes haben unterschiedliche Ausführzeiten, die in „T-Zuständen" gemessen werden. Da außerdem der Z80 auf jeder Maschine mit anderer Geschwindigkeit läuft, muß für die Berechnung der Echtzeitdauer jedes Befehls die Anzahl der T-Zustände durch die Taktfrequenz des Micros geteilt werden. So braucht beispielsweise ein Befehl mit einer Dauer von vier T-Zuständen bei einer Taktfrequenz von zwei Megahertz insgesamt zwei Microsekunden.

Für sehr kleine Verzögerungen läßt sich der NOP-Befehl einsetzen, der auf einem 2-MHz-Micro eine Verzögerung von zwei Microsekunden erreicht. Mehrere NOPs nacheinander verlängern diese Zeit zwar, größere Verzögerungen müssen jedoch über Dummy-Routinen programmiert werden. Die folgenden Befehle erzeugen eine Verzögerung von 27 T-Zuständen:

```
CALL DELAY
RET
```

In diesem Beispiel braucht der CALL-Befehl 17 T-Zustände, und RET benötigt zehn. Bei einer Prozessorgeschwindigkeit von zwei Megahertz beträgt die Verzögerung daher 13,5 Microsekunden. Mit NOP-Befehlen am Anfang der Routine kann diese Zeit aber noch geringfügig verlängert werden.

Noch längere Verzögerungen lassen sich nur über Schleifen programmieren. Das folgende Beispiel lädt ein Register, das über eine Schleife dekrementiert wird. Diese Routine erreicht eine Verzögerung von 99 T-Zuständen (oder 49,5 Microsekunden bei 2 MHz):

BEFEHL	ZEITBEDARF
CALL DELAY	17
(DELAY LOOP)	
LD B,5	7
DEC B	4
JR NZ,LOOP	12 oder 7 wenn „true"

Die Verzögerungsschleife besteht aus drei Befehlen, die mit LD B,5 anfangen. Wie bei dem 6502 verändert sich die Ausführzeit dieser Routine durch den Wert, mit dem das Register geladen wird. Die Taktzyklen für die Ausführung dieses Codes betragen:

C = 24 + (N * 16) — 5

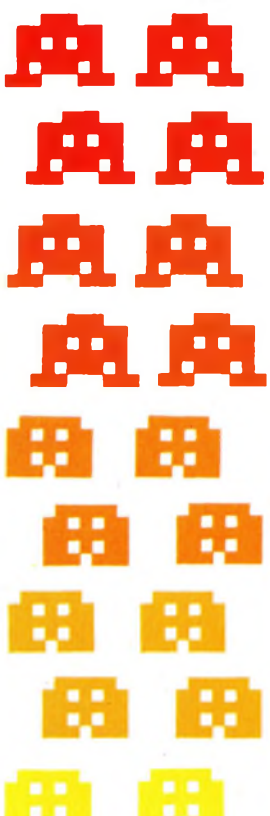
wobei N den Wert darstellt, der in das B-Register geladen wird.

Auch hier können verschachtelte Schleifenzähler eingesetzt werden. Dabei muß jedoch folgendes berücksichtigt werden: Zunächst müssen vor Ausführung einer derartigen Routine alle Register auf den Stapel geschoben werden, damit ihr Inhalt erhalten bleibt. Weiterhin gibt es auf einigen Maschinen Hardware-Interrupts, die den Zeitablauf durcheinanderbringen. Die folgende Routine setzt verschachtelte Schleifen ein (DI und EI schalten die maskierbaren Interrupts aus und wieder ein):

Befehl	Funktion	Zeitbedarf
DI	Interrupt ausschalten	4
PUSH DE	Registerinhalt sichern	11
LD D,nn	Innerer Zähler	7
LD E,nn	Äußerer Zähler	7
CALL OLOOP	Zu äußerem Zähler springen	17
POP DE	Registerinhalt zurückladen	10
EI	Interrupt einschalten	4
(OLOOP)		
DEC E	Außenschleife dekrementieren	4
RET Z	Ende falls Null	11 oder 15
(ILOOP)		
DEC D	Innenschleife dekrementieren	4
JP Z,OLOOP	Springe falls D = Null	10
JP ILOOP	Sonst weiter mit Innenschleife	10

Eine Erhöhung von E verlängert die Verzögerung. Die Routine endet, wenn das E-Register beim Dekrementieren Null enthält. Erreicht der Zähler der inneren Schleife den Wert Null, während der der äußeren Schleife über Eins liegt, wird der innere Schleifenzähler automatisch mit 255 initialisiert. Bevor die Steuerung an die äußere Schleife zurückgegeben wird, muß die innere Schleife bis Null heruntergezählt sein.

In Spielprogrammen sind Verzögerungen des Maschinencodes immer dann notwendig, wenn sich auf dem Bildschirm ein Objekt bewegt, auf das der Spieler reagieren muß. Ein klassisches Beispiel dafür ist das Spiel „Space Invaders". Ohne Zeitverzögerung würden die Bewegungen der eindringenden Aliens zu schnell ablaufen. Durch sorgfältig programmierte Bewegungsverzögerungen wird eine Spielsteuerungüberhaupt erst möglich.



● Die maximalen und minimalen Werte der X-Koordinate. Bei Umrissen mit spitzen Winkeln kann sich Fillsub durchaus über eine Grenzlinie hinausbewegen, wenn der gesetzte Grenzwert noch nicht erreicht ist. Setzen Sie in diesem Fall die Grenzwerte etwas weiter nach innen. Achten Sie darauf, daß die X-Koordinate des Anfangspunktes und die Grenzwerte von X das Format Hi-Byte/Lo-Byte haben (siehe Beispielprogramm).

Obwohl Fillsub die anderen Routinen (Plotsub, Linesub und Circsub) nicht direkt anspricht, lädt das Beispielprogramm diese drei Programme, um die Umrisse zeichnen zu können, die Fillsub dann füllt.

Ladeprogramm für Plotsub/II

Hier ist eine angepaßte Version der ersten Plotsub-Routine. Legen Sie damit auf Cassette oder Diskette die neue Objectcodedatei „PLOTSUB.HEX" ab.

```
10 FORI=49408TO49408+314
20 READA:POKEI,A:S=S+A:NEXT
30 READCC:IFCC<>STHENPRINT"CHECKSUM ERROR"
100 DATA1,0,3,6,0,5,0,0,6,5,5,69,38,2
110 DATA72,138,72,152,72,173,0,193,240
120 DATA83,169,0,133,251,169,4,133,252
130 DATA162,3,160,0,173,2,193,145,251
140 DATA136,208,251,230,252,202,48,8
150 DATA208,244,145,251,160,231,208
160 DATA238,173,1,193,240,24,169,0,133
170 DATA251,169,32,133,252,162,32,160
180 DATA0,169,0,145,251,136,208,251
190 DATA230,252,202,208,246,173,24,208
200 DATA41,240,9,8,141,24,208,173,17
210 DATA208,9,32,141,17,208,76,125,193
220 DATA173,24,208,41,240,9,4,141,24
230 DATA208,173,17,208,41,223,141,17
240 DATA208,104,168,104,170,104,96,72
250 DATA138,72,152,72,173,4,193,141,7
260 DATA193,173,3,193,41,248,141,6,193
270 DATA173,3,193,41,7,141,8,193,173,5
280 DATA193,41,7,141,10,193,162,3,78,5
290 DATA193,202,208,250,173,5,193,141
300 DATA9,193,169,0,141,11,193,141,12
310 DATA193,162,5,173,11,193,24,109,9
320 DATA193,141,11,193,202,208,243,162
330 DATA6,14,12,193,14,11,193,144,3
340 DATA238,12,193,202,208,242,173,11
350 DATA193,24,109,6,193,141,11,193
360 DATA173,12,193,109,7,193,141,12
370 DATA193,173,11,193,24,105,0,141,11
380 DATA193,173,12,193,105,32,141,12
390 DATA193,173,11,193,24,109,10,193
400 DATA141,11,193,173,12,193,105,0
410 DATA141,12,193,173,11,193,133,251
420 DATA173,12,193,133,252,169,1,141
430 DATA13,193,56,169,7,237,8,193,240
440 DATA7,170,14,13,193,202,208,248
450 DATA160,0,177,251,13,13,193,145
460 DATA251,76,125,193
470 DATA37523:REM*CHECKSUM*
```

In Zeile 15 zeigt DN=8 an, daß die Objectcodedateien (Plotsub.Hex etc.) von der Diskette geladen werden sollen. Ändern Sie dies in DN=1 um, wenn Sie den Code von Cassetten laden wollen. Legen Sie nun entweder eine Cassette mit allen drei Objectcodedateien an (in der Reihenfolge, die von Zeile 20 bis 30 aufgeführt ist), oder setzen Sie den folgenden Code als Zeile 22, 26 und 28 ein:

INPUT "CASSETTE WECHSELN UND RETURN DRUECKEN"; A$

Fillsub-Beispielprogramm

```
10 REM **** FILLSUB DEMO PROGRAMM ****
15 DN=8 REM FUR CASSETTE DN=1
20 IFA=0THENA=1 LOAD"PLOTSUB.HEX",DN,1
25 IFA=1THENA=2 LOAD"LINESUB.HEX",DN,1
27 IFA=2THENA=3 LOAD"CIRCSUB.HEX",DN,1
30 IFA=3THENA=4 LOAD"FILLSUB.HEX",DN,1
40 GOSUB1000 REM SET HIRES
50 REM **** DRAW TRIANGLE ****
60 XA=100 YA=150 XB=300 YB=160 XC=170 YC=20
80 X1=XA Y1=YA X2=XB Y2=YB GOSUB2000
90 X1=XC Y1=YC GOSUB2000
100 X2=XA Y2=YA GOSUB2000
102 REM **** DRAW CIRCLE ****
103 XC=60 YC=60 R=50 GOSUB4000
120 REM **** FILL TRIANGLE ****
130 XS=170 YS=130 REM START POINTS
140 MIN=100 MAX=299 REM LIMITS
150 GOSUB3000
161 REM **** FILL CIRCLE ****
162 XS=60 YS=60 REM START POINT
163 MIN=10 MAX=109
164 GOSUB3000
```

Fillsub-Beispielprogramm (Fortsetzung)

```
200 GETA$:IFA$=""THEN200 REM AWAIT KEYPRESS
210 POKE49408,0:SYS49422 REM RESET SCREEN
220 PRINTCHR$(147) REM CLEAR SCREEN
225 PRINT"END OF ROUTINE"
230 END
1000 REM **** SET HIRES ****
1010 POKE49408,1:POKE49409,1
1020 POKE49410,7
1030 SYS49422
1040 RETURN
2000 REM **** LINESUB ****
2010 MHI=INT(X1/256):MLO=X1-256*MHI
2020 NHI=INT(X2/256):NLO=X2-256*NHI
2030 POKE49920,MLO:POKE49921,MHI
2040 POKE49922,NLO:POKE49923,NHI
2050 POKE49924,Y1:POKE49925,Y2
2060 SYS 49934
2070 RETURN
3000 REM **** FILLSUB ****
3010 SH=INT(XS/256):SL=XS-SH*256
3020 HAX=INT(MAX/256):LAX=MAX-256*HAX
3030 HIN=INT(MIN/256):LIN=MIN-256*HIN
3040 POKE50955,SL:POKE50956,SH
3050 POKE50957,YS
3060 POKE50958,LIN:POKE50959,HIN
3070 POKE50960,LAX:POKE50961,HAX
3080 SYS50967
3090 RETURN
4000 REM **** CIRCSUB ****
4010 CHI=INT(XXC/256):CLO=XC-256*CHI
4020 POKE50497,CLO:POKE50498,CHI
4030 POKE50499,YC:POKE50500,R
4040 SYS 50521
4050 RETURN
```

Ladeprogramm für Fillsub

```
10 REM **** BASIC LOADER FOR FILLSUB ****
20 FORI=50944 TO 51375
30 READA:POKEI,A:CC=CC+A:NEXT
40 READA:IFCC<>A THEN PRINT"CHECKSUM ERROR":END
100 DATA11,0,6,8,0,3,6,5,136,39,16,60
110 DATA0,60,10,0,109,0,0,1,10,0,16
120 DATA173,11,199,141,20,199,173,12
130 DATA199,141,21,199,172,13,199,169
140 DATA1,141,18,199,141,19,199,140,5
150 DATA193,173,20,199,141,3,193,173
160 DATA21,199,141,4,193,32,131,193
170 DATA173,19,199,208,8,200,192,200
180 DATA240,19,76,82,199,136,192,0,144
190 DATA11,32,246,199,173,22,199,208,3
200 DATA76,46,199,173,18,199,208,31
210 DATA173,20,199,56,233,1,141,20,199
220 DATA173,21,199,233,0,141,21,199
230 DATA205,15,199,208,65,173,20,199
240 DATA205,14,199,208,57,96,173,20
250 DATA199,24,105,1,141,20,199,173,21
260 DATA199,105,0,141,21,199,205,17
270 DATA199,208,34,173,20,199,205,16
280 DATA199,208,26,173,11,199,141,20
290 DATA199,173,12,199,141,21,199,172
300 DATA13,199,169,0,141,19,199,141,18
310 DATA199,76,46,199,173,19,199,208
320 DATA28,136,32,246,199,173,22,199
330 DATA208,4,200,76,191,199,238,19
340 DATA199,136,32,246,199,173,22,199
350 DATA208,247,76,46,199,200,32,246
360 DATA199,173,22,199,208,4,136,76
370 DATA219,199,206,19,199,200,32,246
380 DATA199,173,22,199,208,247,76,46
390 DATA199,72,138,72,152,72,140,2,199
400 DATA173,20,199,141,0,199,173,21
410 DATA199,141,1,199,173,1,199,141,4
420 DATA199,173,0,199,41,248,141,3,199
430 DATA173,0,199,41,7,141,5,199,173,2
440 DATA199,41,7,141,7,199,162,3,78,2
450 DATA199,202,208,250,173,2,199,141
460 DATA6,199,169,0,141,8,199,141,9
470 DATA199,162,5,173,8,199,24,109,6
480 DATA199,141,8,199,202,208,243,162
490 DATA6,14,8,199,46,9,199,202,208
500 DATA247,173,8,199,24,109,3,199,141
510 DATA8,199,173,9,199,109,4,199,141
520 DATA9,199,173,8,199,24,105,0,141,8
530 DATA199,173,9,199,105,32,141,9,199
540 DATA173,8,199,24,109,7,199,133,251
550 DATA173,9,199,105,0,133,252,169,1
560 DATA141,10,199,56,169,7,237,5,199
570 DATA240,7,170,14,10,199,202,208
580 DATA50,160,0,177,251,45,10,199
590 DATA141,22,199,104,168,104,170,104
600 DATA96
610 DATA50785:REM*CHECKSUM*
```

Assemblerprogramm

```
                ;
                ;********************
                ;********************
                ;**                **
                ;**  FILLSUB 64    **
                ;**                **
                ;********************
                ;********************
                ;
                ;**** PLOTSUB VALUES ****
                ;
                PLTSUB = $C183
                XLO    = $C103
                XHI    = $C104
                YLO    = $C105
                MPBLO  = $60
                MPBHI  = $20
                PTR    = $FB
                *      = $C700
                ;
                ;**** FILLSUB VARIABLES ****
                ;
                PXLO   *=*+1
                PXHI   *=*+1
                PYLO   *=*+1
                PHBLO  *=*+1
                PHBHI  *=*+1
                PREMX  *=*+1
                PYBYTE *=*+1
                PREMY  *=*+1
                PROWLO *=*+1
                PROWHI *=*+1
                PBPOS  *=*+1
                ;
                XSTLO  *=*+1
                XSTHI  *=*+1
                YST    *=*+1
                XMINLO *=*+1
                XMINHI *=*+1
                XMAXLO *=*+1
                XMAXHI *=*+1
                RTFLAG *=*+1
                UPFLAG *=*+1
                PXLO   *=*+1
                PXHI   *=*+1
                PTFLAG *=*+1
                ;
                ;**** TRANS START PT TO PX,PY ****
                ;
AD 0B C7        START   LDA XSTLO
8D 14 C7                STA PXLO
AD 0C C7                LDA XSTHI
8D 15 C7                STA PXHI
AC 0D C7                LDY YST
                ;
                ;**** SET FLAGS ****
                ;
A9 01                   LDA #$01
8D 12 C7                STA RTFLAG
8D 13 C7                STA UPFLAG
                ;
                ;**** PLOT POINT ****
                ;
                NEXT
8C 05 C1                STY YLO
AD 14 C7                LDA PXLO
8D 03 C1                STA XLO
AD 15 C7                LDA PXHI
8D 04 C1                STA XHI
20 83 C1                JSR PLTSUB
                ;
                ;**** INC,/ DEC Y COORD ****
                ;
AD 13 C7                LDA UPFLAG
D0 08                   BNE DECRY
C8                      INY
C0 C8                   CPY #$C8        ;HAS Y REACHED MAX
F0 13                   BEQ NEXLIN
4C 52 C7                JMP TESTPT
                DECRY
88                      DEY
C0 00                   CPY #$00        ;HAS Y REACHED MIN
90 0B                   BCC NEXLIN
                TESTPT
20 F6 C7                JSR POINT       ;IS POINT LIT ?
AD 16 C7                LDA PTFLAG
D0 03                   BNE NEXLIN      ;IF YES BRANCH
4C 2E C7                JMP NEXT
                ;
                ;**** START NEXT LINE ****
                ;
                NEXLIN
AD 12 C7                LDA RTFLAG
D0 1F                   BNE INCRX       ; TEST RIGHT FLAG
AD 14 C7                LDA PXLO
38                      SEC
E9 01                   SBC #$01
8D 14 C7                STA PXLO        ;DEC X LOBYTE
AD 15 C7                LDA PXHI
E9 00                   SBC #$00
8D 15 C7                STA PXHI        ;DEC X HIBYTE

CD 0F C7                CMP XMINHI
D0 41                   BNE MOVE
AD 14 C7                LDA PXLO        ;HAS X REACHED XMIN ?
CD 0E C7                CMP XMINLO
D0 3B                   BNE MOVE
60                      RTS             ;END OF ROUTINE
                INCRX
AD 14 C7                LDA PXLO
18                      CLC
69 01                   ADC #$01        ;INC X LOBYTE
8D 14 C7                STA PXLO
AD 15 C7                LDA PXHI
69 00                   ADC #$00        ;INC X HIBYTE
8D 15 C7                STA PXHI

CD 11 C7                CMP XMAXHI
D0 22                   BNE MOVE
AD 14 C7                LDA PXLO        ;HAS X REACHED XMAX ?
CD 10 C7                CMP XMAXLO
D0 1A                   BNE MOVE

AD 0B C7                LDA XSTLO
8D 14 C7                STA PXLO
AD 0C C7                LDA XSTHI       ;RESET START POINT
8D 15 C7                STA PXHI
AC 0D C7                LDY YST

A9 00                   LDA #$00
8D 13 C7                STA UPFLAG
8D 12 C7                STA RTFLAG      ;RESET FLAGS
4C 2E C7                JMP NEXT
                ;
                ;**** FIND START OF NEXT LINE ****
                ;
                MOVE
AD 13 C7                LDA UPFLAG
D0 10                   BNE DOWN        ;TEST UPFLAG
88                      DEY             ;Y=Y-1
                AGAIN1
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 04                   BNE CONT1
C8                      INY
4C BF C7                JMP AGAIN1
                CONT1
EE 13 C7                INC UPFLAG      ;SET TO ONE
                AGAIN2
88                      DEY
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 F7                   BNE AGAIN2
4C 2E C7                JMP NEXT
                ;
                DOWN
C8                      INY             ;Y=Y+1
                AGAIN3
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 04                   BNE CONT2
88                      DEY
4C DB C7                JMP AGAIN3
                CONT2
CE 13 C7                DEC UPFLAG      ;SET TO ZERO
                AGAIN4
C8                      INY
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 F7                   BNE AGAIN4
4C 2E C7                JMP NEXT
                ;**** END OF MAIN PROGRAM ****
                ;
                ;**** TEST POINT SUBROUTINE ****
                ;
                POINT
48                      PHA
8A                      TXA
48                      PHA             ;PUSH REGS ONTO STACK
98                      TYA
48                      PHA
                ;
8C 02 C7                STY PYLO
AD 14 C7                LDA PXLO
8D 00 C7                STA PXLO;TRANSFER COORDS
AD 15 C7                LDA PXHI
8D 01 C7                STA PXHI
                ;
                ;**** CALCULATE ADDRESS OF POINT ****
                ;
AD 01 C7                LDA PXHI
8D 04 C7                STA PHBHI
AD 00 C7                LDA PXLO
29 F8                   AND #$F8
8D 03 C7                STA PHBLO
AD 00 C7                LDA PXLO
29 07                   AND #$07
8D 05 C7                STA PREMX

AD 02 C7                LDA PYLO
29 07                   AND #$07
8D 07 C7                STA PREMY

A2 03                   LDX #$03
                SHIFT
4E 02 C7                LSR PYLO
CA                      DEX
10 FA                   BNE SHIFT
AD 02 C7                LDA PYLO
8D 06 C7                STA PYBYTE

A9 00                   LDA #$00
8D 09 C7                STA PROWLO
8D 0A C7                STA PROWHI
A2 05                   LDX #$05
                FIVE
18                      CLC
6D 06 C7                ADC PYBYTE
8D 08 C7                STA PROWLO
CA                      DEX
D0 F3                   BNE FIVE
A2 06                   LDX #$06
                MULT
0E 08 C7                ASL PROWLO
2E 0A C7                ROL PROWHI
CA                      DEX
D0 F7                   BNE MULT

AD 08 C7                LDA PROWLO
18                      CLC
6D 03 C7                ADC PHBLO
8D 08 C7                STA PROWLO
AD 09 C7                LDA PROWHI
6D 04 C7                ADC PHBHI
8D 09 C7                STA PROWHI

AD 08 C7                LDA PROWLO
18                      CLC
69 00                   ADC #MPBLO
8D 08 C7                STA PROWLO
AD 09 C7                LDA PROWHI
69 20                   ADC #MPBHI
8D 09 C7                STA PROWHI

AD 08 C7                LDA PROWLO
18                      CLC
6D 07 C7                ADC PREMY
85 FB                   STA PTR
AD 09 C7                LDA PROWHI
69 00                   ADC #$00
85 FC                   STA PTR+1

A9 01                   LDA #$01
8D 0A C7                STA PBPOS
38                      SEC
A9 07                   LDA #$07
ED 05 C7                SBC PREMX
F0 07                   BEQ BITON
AA                      TAX
                POWER
0E 0A C7                ASL PBPOS
CA                      DEX
D0 FA                   BNE POWER
                ;
                ;**** TEST FOR BIT ON ****
                ;
                BITON
A0 00                   LDY #$00
B1 FB                   LDA (PTR),Y     ;LOAD ADDRESS CONTENTS
2D 0A C7                AND PBPOS
8D 16 C7                STA PTFLAG      ;STORE RESULT
                ;
68                      PLA
A8                      TAY
68                      PLA             ;PULL REGS OFF STACK
AA                      TAX
68                      PLA
                ;
60                      RTS
```

Der Maschinencode des 6809

Wir setzen unseren Maschinencodekurs mit einer Artikelserie über die Assemblersprache des 6809-Prozessors fort, der unter anderem die Heimcomputer Dragon und Tandy Color steuert. Zunächst wird untersucht, welche Rolle die Register dieses Prozessors spielen.

Microprozessoren haben im allgemeinen drei Hauptfunktionsbereiche: die Register (interne Speicher des Prozessors), die ALU (Arithmetik- und Logikeinheit, die einfache mathematische Vorgänge ausführt) und eine Steuereinheit, die veranlaßt, daß alle Vorgänge in der richtigen Reihenfolge und zur richtigen Zeit ablaufen.

Auf der untersten Funktionsebene reagieren Microprozessoren auf elektrische Impulse (die sie über externe Anschlußkontakte erhalten), indem sie entweder den Zustand ihrer internen Register ändern oder andere Impulse aussenden oder entgegennehmen. Wenn wir die Anwesenheit eines Stromimpulses als Eins ansehen und die Abwesenheit als Null, lassen sich diese Signale – die entweder zwischen Prozessor und Speicher oder nur im Prozessorinneren hin- und hergehen – als Zahlen eines binären Zahlensystems darstellen. Prozessoren werden programmiert, indem man sie mit einer Folge von Zahlen (bzw. Befehlen) versorgt, die sie dann wie gewünscht bearbeiten bzw. ausführen.

Acht-Bit-Prozessoren wie der 6809 können Binärzahlen mit einer Länge von acht Bits senden und empfangen. Sie umfassen damit die Dezimalzahlen 0 bis 255. Viele dieser Zahlen beziehen sich auf Speicheradressen. Diese werden auf den meisten Acht-Bit-Prozessoren als 16-Bit-Zahlen angegeben. Sie adressieren Speicherstellen von 0 bis 65535. Diese 16-Bit-Adressen verarbeitet der Prozessor natürlich nur in zwei Teilen zu je acht Bit.

Vier Hauptregister

Die Register eines Prozessors können unterschiedliche Formate haben. Einige sind für den internen Gebrauch reserviert und dem Programmierer nicht zugänglich. Der 6809 hat vier Hauptregister.

Am häufigsten wird der *Akkumulator* angesprochen, in dem auch die meisten Daten gespeichert und bearbeitet werden. Beispielsweise addiert der ADD-Befehl den Inhalt einer speziellen Speicherstelle auf die Daten, die sich gerade im Akkumulator befinden. Der Ergebniswert wird so in diesem wichtigen Register „akkumuliert" (angesammelt).

Mit dem *Indexregister* lassen sich Adressen verändern, so daß dann Tabellen und Daten leicht schrittweise angesprochen werden können. Ein Befehl, der sich über die *indizierte Adressierung* auf eine Speicherstelle bezieht, addiert den Inhalt des Indexregisters auf die angegebene Basisadresse und spricht so die *aktuelle Adresse* an. Für die schrittweise Bearbeitung von Datentabellen braucht nur die *Basisadresse* (das erste Datenelement der Tabelle) angegeben und das Indexregister inkrementiert zu werden. Da Indexregister normalerweise Adressen enthalten, haben sie im allgemeinen eine Länge von 16 Bits.

Der *Stack Pointer* enthält die oberste Adresse des *Stacks*, der sich besonders zur schnellen Zwischenspeicherung von Daten eignet. Der Stack wird häufig zur Speicherung des internen Prozessorregisters eingesetzt (etwa beim Aufruf von Unterroutinen), dessen Daten später wieder zurückgeladen werden müssen. Es können beliebig viele Register auf den Stack geschoben und wieder abgezogen werden. Der Stack Pointer enthält dabei die Speicheradresse, auf der die letzte Information abgelegt wurde. Da Stack Pointer sich auf Speicheradressen beziehen, sind sie im allgemeinen ebenfalls 16 Bits lang.

Das vierte Register – der *Programmzähler* – führt seine Funktion fast immer automatisch aus. Er enthält jeweils die Adresse des nächsten Befehls. Der Prozessor spricht diese Speicherstelle an, holt sich ihren Inhalt, interpretiert die Bedeutung und führt den Befehl aus. Normalerweise wird der Programmzähler bei jeder ausgeführten Anweisung automatisch inkrementiert. Verändert man aber seinen Inhalt (durch Addition oder Subtraktion oder durch Speicherung eines neuen Wertes), dann ändert sich auch der Programmablauf. Obwohl dieser Vorgang dem GOTO-Befehl ähnelt, werden auf Maschinenebene neue Adressen mit dem Sprungbefehl (JMP) angegeben und aktuelle Adressen mit dem Verzweigungsbefehl (BRA) verändert.

Die fünfte Registerart funktioniert völlig anders. Das *Condition Code Register* läßt sich als eine Reihe einzelner Bits beschreiben, die In-

Die 6809-Geräte mit der größten Verbreitung sind die Heimcomputer Dragon 32 und 64 und der Tandy Color Computer. Außerdem gibt es eine große Zahl von 6809-Entwicklungsgeräten, die in Universitäten und technischen Hochschulen im Einsatz sind.

formationen über bestimmte Zustände des Prozessors enthalten. Beispielsweise zeigt nach einem Prozessorvorgang eins dieser Bits an, ob das in einem internen Register abgelegte Ergebnis Null ist. So lassen sich Wertetabellen schrittweise abarbeiten, indem die Gesamtzahl der Werte in ein Register geladen und nach jedem Bearbeitungsvorgang von dieser Zahl eine Eins abgezogen wird. Erreicht dieses Register den Wert Null, dann enthält das Bit des Condition Codes die Information, daß alle Tabellenwerte abgearbeitet sind und der nächste Befehl angesprochen werden muß. Mit dieser Befehlsart lassen sich Bedingungen (IF-Anweisungen) und Schleifen (FOR . . .NEXT, WHILE . . . WEND und REPEAT . . .UNTIL) konstruieren.

Viele Prozessoren enthalten eine *Zero Page*, die normalerweise aus den ersten 256 Speicherstellen (Hex 0000 bis 00FF) besteht und mit Acht-Bit-Adressen angesprochen wird. Befehle mit diesem Adreßformat sind kürzer und werden auch schneller ausgeführt. Der Prozessor 6809 verfügt über ein *Direct Page Register* im Acht-Bit-Format, das die zusätzlichen acht Adreßbits der Zero-Page-Adressen enthält. Da sich der Wert dieses Registers verändern läßt, kann die Zero Page in einen beliebigen Speicherbereich verlegt oder sogar mehrfach eingerichtet werden.

Maschinencodeprogramme bestehen aus Folgen von Anweisungen, die sich aus Daten und Adressen zusammensetzen. Einige Programmierer können diese Werte direkt in numerischen Werten angeben – diese Methode ist jedoch für die meisten zu kompliziert. Maschinencodeprogramme lassen sich weitaus einfacher in der Assemblersprache schreiben, die die Befehle durch mnemotische Kürzel und die Adressen und Daten durch Labels ersetzt. Wenn wir beispielsweise die Daten einer Speicherstelle in den Akkumulator laden möchten, können wir

```
STORE FCB 0
```

schreiben, um im Speicher einen Platz zu reservieren, auf den wir uns mit STORE beziehen und auf dem nun eine Null gespeichert ist. FCB ist eigentlich kein Befehl, sondern eine Anweisung, die dem Übersetzungsprogramm (das den Quellentext der Assemblersprache in den Maschinencode überträgt) mitteilt, daß es für das Wort STORE eine bestimmte Adresse einsetzen soll. Der dort gespeicherte Wert läßt sich mit

```
LDA STORE
```

in den Akkumulator laden.

Programme in Assemblersprache müssen vor ihrem Ablauf von einem Spezialprogramm – dem *Assembler* – übersetzt werden. Assembler sind nicht allzu kompliziert, da eine Assembleranweisung dem daraus entstehenden Maschinencode fast direkt entspricht. Die Übersetzung tauscht nur die Kürzel gegen Zahlen aus und setzt statt der Namen die entsprechenden Werte und Adressen ein.

Register ziehen

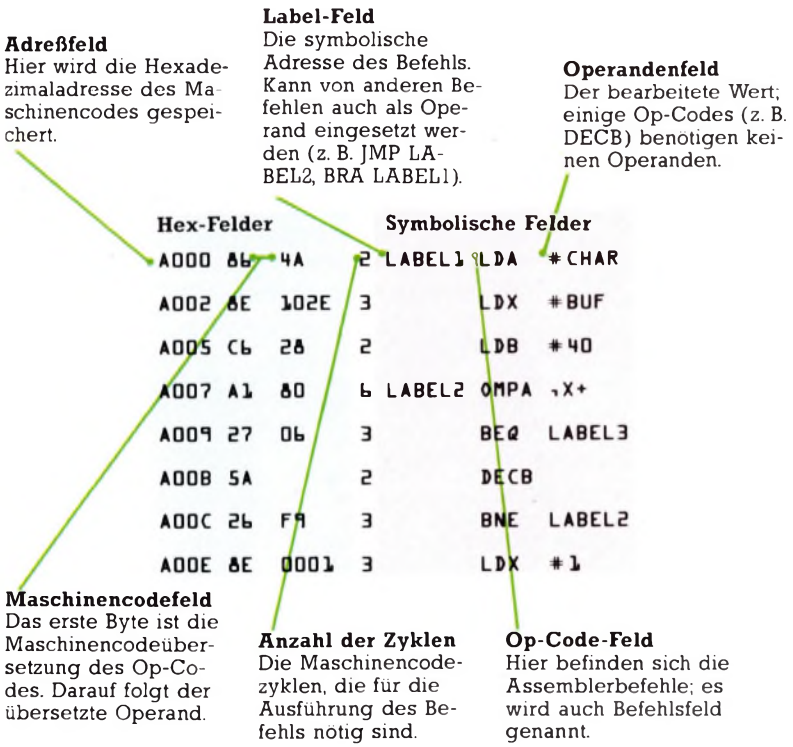
Bei der Vorstellung des 6809 wurden zunächst die allgemeinen Funktionen des Prozessorregisters erläutert. In diesem Artikel untersuchen wir nun im einzelnen, welche Aufgaben die Register des 6809 ausführen.

Register sind Speicherstellen, die sich im Prozessorchip befinden. Die Bewegung von Daten vom Arbeitsspeicher in diese Register und umgekehrt steuert den Ablauf der Assemblersprache. Einige Register – hauptsächlich die Adreßspeicher – haben eine Länge von 16 Bits, während die anderen acht Bits lang sind und eine Vielzahl verschiedener Aufgaben ausführen.

- **Indexregister** verändern die Adressen eines Programms.
- **Stapelzeiger** (Stack Pointer) adressieren einen bestimmten Bereich des Hauptspeichers, der für schnelle Zwischenspeicherung eingesetzt wird.
- Der **Befehlszähler** enthält die Adresse des nächsten Befehls. Er kann zur Programmsteuerung verändert werden. Der Befehlszähler entspricht etwa der GOTO-Anweisung.
- Die **Akkumulatoren** werden am häufigsten für mathematische Abläufe eingesetzt.
- Das **Conditions-Code-Register** enthält eine Reihe Flags, die den Zustand des Prozessors anzeigen (zum Beispiel ob das Ergebnis des letzten Vorgangs Null ist). Die Flags können für Verzweigungen oder Schleifen eingesetzt werden und geben der Assemblersprache damit ein Gegenstück zur IF...THEN-Struktur.

Der 6809 besitzt all diese Register. Da er eine Weiterentwicklung des 6800 von Motorola ist (wie auch der 6502), bestehen viele Ähnlichkeiten zwischen den Assemblersprachen beider Prozessoren. Die beiden Versionen sind jedoch nicht kompatibel. Code, der für den einen Prozessor geschrieben wurde, funktioniert nicht auf dem anderen. Viele 6800-Programme sind jedoch im Quellentext kompatibel, das heißt, ein für den 6800 geschriebenes Assemblerprogramm, das für den 6809 assembliert wird, kann durchaus funktionieren. Diese Art der Kompatibilität gibt es nicht bei dem 6502. Durch die Ähnlichkeit der Prozessoren kann jedoch jeder, der mit dem 6502 vertraut ist, leicht dessen Assemblerprogramme in eine entsprechende 6809-Version für folgende Register umsetzen:

- Zwei Acht-Bit-Akkumulatoren mit den Bezeichnungen A und B. Da es als Acht-Bit-Register keine wesentliche Unterschiede zwischen ihnen gibt, können sie beide als Akkumulator eingesetzt werden. Dies bringt den Vorteil, daß die Werte des einen Akkumulators erhalten bleiben, während der andere arbeitet. Beide Acht-Bit-Akkumulatoren lassen sich jedoch auch zusammen als ein 16-Bit-Akkumulator direkt für die Ausführung von 16-Bit-Arithmetik einsetzen. Wegen dieser Eigenschaft wird der 6809 oft als Pseudo-16-Bit-Prozessor bezeichnet. Diese Bezeichnung ist jedoch nicht korrekt. Es ist besser, ihn als hochentwickelten Acht-Bit-Prozessor anzusehen. Die Akkumulatoren werden beim gemeinsamen Einsatz als das 16-Bit-Register D bezeichnet.
- Auch zwischen den beiden Indexregistern X und Y gibt es keine wesentlichen Funktionsunterschiede. Da jedoch einige Befehle, die das Y-Register ansprechen, die Länge von zwei Bytes haben (im Gegensatz zu den entsprechenden Ein-Byte-Instruktionen des X-Registers), sollte, wenn möglich, nur das X-Register eingesetzt werden, da die Ausführzeit auf diese Art kürzer ist.
- Der 6809 hat zwei Stack-Pointer, die mit S und U bezeichnet werden. Alle Stack-Vorgänge des Prozessors laufen über S. Obwohl man auch S ansprechen kann, muß sicherge-



Assemblerprogramme haben ein anderes Format als BASIC-Programme. So sehen die von einem Assembler (ein Programm, das die Assemblersprache in den Maschinencode übersetzt) gedruckten Listen erst einmal verwirrend aus. Schlüssel zum Verständnis sind die Spalten (oder Felder). Oft sind nur die Spalten für Label, Op-Code und Operand wichtig, während der Rest ignoriert werden kann.

stellt sein, daß dadurch die Prozessorabläufe nicht gestört werden. Es ist daher sicherer, nur mit U zu arbeiten. Aufgrund der beiden Stack-Pointer eignet sich der 6809 ausgezeichnet für FORTH.

● Der Einsatz der Register X, Y, S und U muß nicht auf ihre vorbestimmte Rolle beschränkt bleiben. So lassen sich S und U auch als Indexregister einsetzen, während alle Register 16-Bit-Zahlen bearbeiten können.

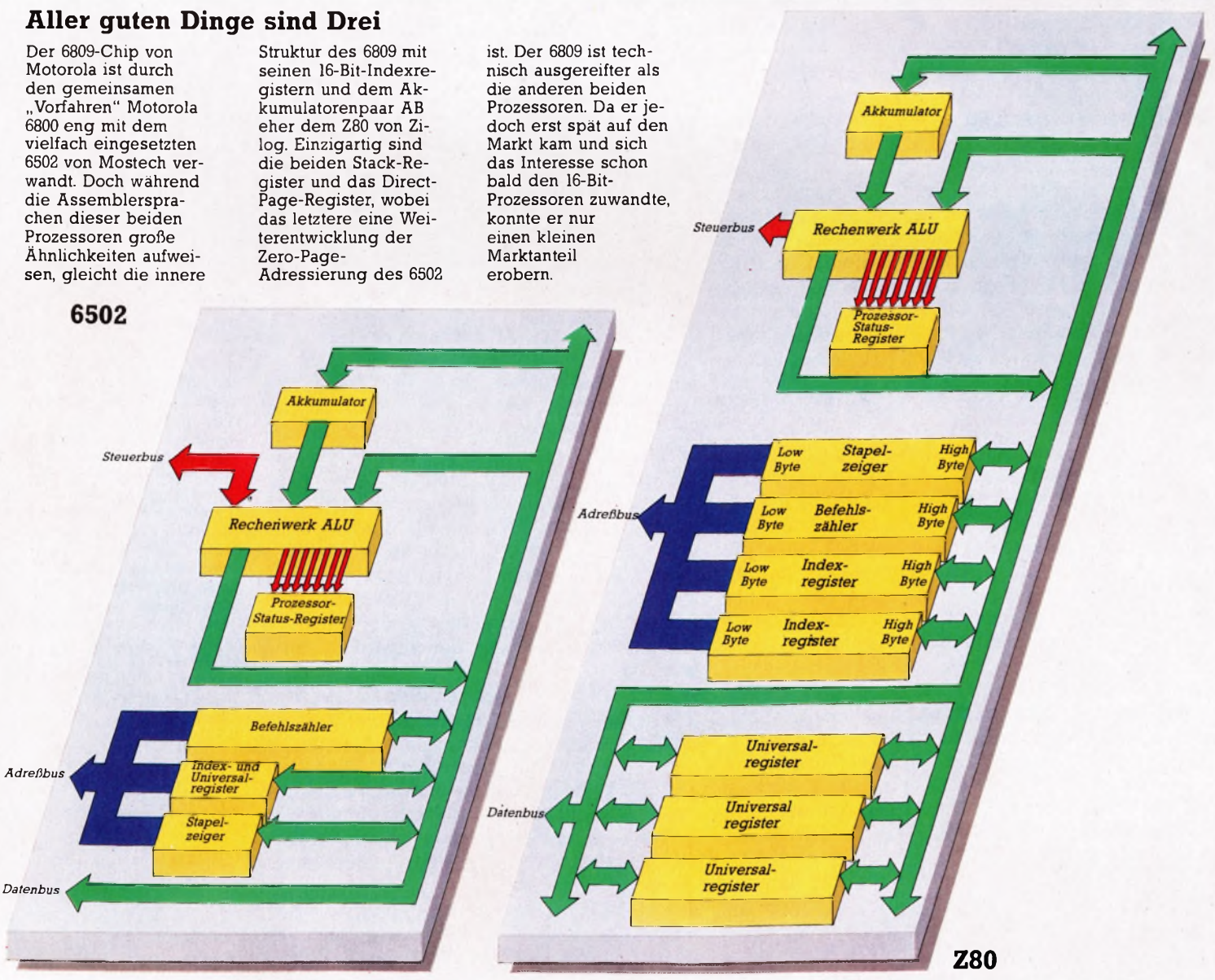
● Der Befehlszähler (PC) ist ein 16-Bit-Register, das vom Prozessor automatisch auf den nächsten Befehl gestellt wird. Die Befehle für Sprünge (JMP) und Verzweigungen (BRA) verändern den Inhalt des Befehlszählers, der auf dem 6809 auch als eine Art Indexregister verwandt werden kann. Wir werden uns später im Kurs noch ausführlich mit dieser Möglichkeit beschäftigen, da durch den Einsatz von Adressen relativ zum Befehlszähler ein Code geschrieben werden kann, der nicht von absoluten Adressen abhängig ist. Mit anderen Worten: Entsprechend programmierte Module können in einem beliebigen Bereich des Speichers abgelegt werden und dort ohne weiteres funktionieren.

● Das Condition-Code-Register (CC) hat acht Bits, die unabhängig voneinander als Flags eingesetzt werden und den Status des Prozessors anzeigen.

Einige Befehle bewegen nur Daten von einem Register zum anderen. Für ein Beispiel wollen wir mit den Assembleranweisungen FCB und FDB eine Anzahl Speicherstellen reservieren. Wenn der Assembler bei einer Übersetzung des Quellentextes in den Maschinencode auf diese Anweisungen trifft, wird dadurch der entsprechende Speicherbereich für das Programm reserviert. FCB und FDB sind keine Prozessorbefehle, und da sie den Op-Codes der Assemblersprache ähnlich sind, werden sie oft Pseudo-Ops genannt. Die Bezeichnung „Assembleranweisungen" bezieht sich jedoch deutlicher auf ihre Funktion,

Aller guten Dinge sind Drei

Der 6809-Chip von Motorola ist durch den gemeinsamen „Vorfahren" Motorola 6800 eng mit dem vielfach eingesetzten 6502 von Mostech verwandt. Doch während die Assemblersprachen dieser beiden Prozessoren große Ähnlichkeiten aufweisen, gleicht die innere Struktur des 6809 mit seinen 16-Bit-Indexregistern und dem Akkumulatorenpaar AB eher dem Z80 von Zilog. Einzigartig sind die beiden Stack-Register und das Direct-Page-Register, wobei das letztere eine Weiterentwicklung der Zero-Page-Adressierung des 6502 ist. Der 6809 ist technisch ausgereifter als die anderen beiden Prozessoren. Da er jedoch erst spät auf den Markt kam und sich das Interesse schon bald den 16-Bit-Prozessoren zuwandte, konnte er nur einen kleinen Marktanteil erobern.

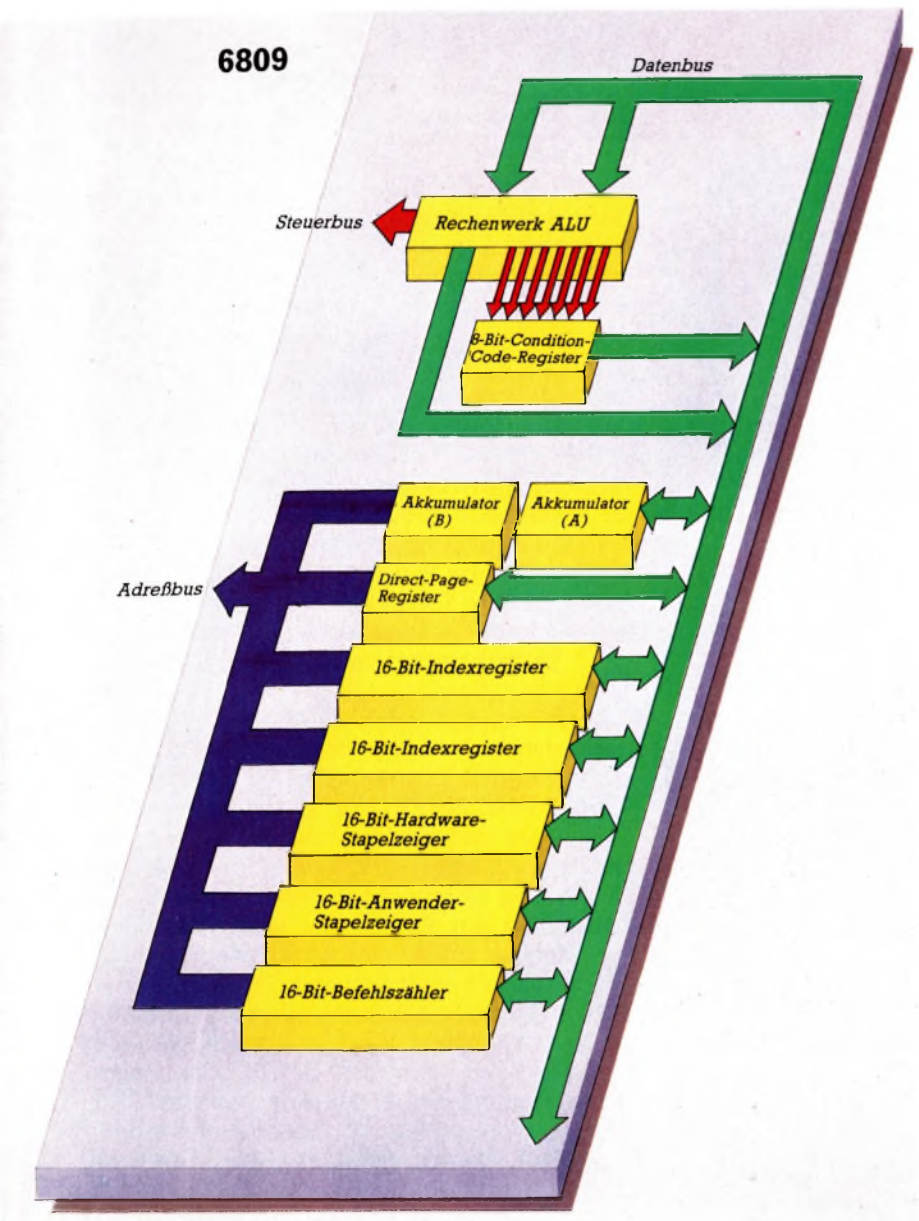


die Steuerung des Assemblers. Wenn die anweisungen mit Namen (Labels) versehen werden, übersetzt der Assembler diese Labels in die entsprechenden Adressen.

```
NUM1 FCB 0      reserviert ein Byte mit dem
                Namen NUM1 und setzt es
                auf den Anfangswert 0.
NUM2 FCB 0      gleicher Vorgang wie oben.
NUM3 FCB #A93B  reserviert zwei Bytes für
                die 16-Bit-Zahl #A93B
                (# zeigt in 6809-Assem-
                blern im allgemeinen das
                Hexadezimalformat einer
                Zahl an).
```

Die folgenden Befehle laden die in diesen Adressen gespeicherten Werte in die entsprechenden Register:

```
LDA NUM1    lädt eine Acht-Bit-Zahl aus
            NUM1 in den Akkumulator A.
LDB NUM2    lädt wie zuvor NUM2 in den
            Akkumulator B.
```




```
LDX NUM3
LDY NUM3    Diese Befehle laden die in
LDS NUM3    NUM3 gespeicherte 16-Bit-Zahl
LDU NUM3    entsprechend in die Register X,
LDD NUM3    Y, S, U und D.
```

Auf die gleiche Weise kann der Acht- oder 16-Bit-Inhalt eines Registers wieder im Speicher abgelegt werden:

```
STA NUM1
STB NUM2
STX NUM3
STY NUM3
STS NUM3
STU NUM3
STD NUM3
```

Beachten Sie, daß der Wert von NUM1 beim Laden des Akkumulators nur kopiert, aber nicht verändert wird. Die Speichervorgänge funktionieren auf die gleiche Weise.

Die Inhalte zweier Register lassen sich mit dem Befehl EXG gegeneinander austauschen (vorausgesetzt, ihre Größe ist identisch):

```
EXG A,B vertausche den Inhalt der
        Register A und B
EXG X,S vertausche den Inhalt der
        Register X und S
```

Der Inhalt eines Registers kann auch in ein anderes übertragen werden: TFR Y,U kopiert den Inhalt von Y nach U. Auch hier müssen beide Register die gleiche Länge haben – acht Bit oder 16 Bit.

Nehmen wir für ein Beispielprogramm den Befehl ADD, der den Inhalt einer Speicherstelle auf den Inhalt eines der Akkumulatoren addiert. Er hat folgendes Format:

```
ADDA NUM1   bedeutet "addiere den Inhalt
            der Speicherstelle NUM1 zum
            A-Register, so daß das A-Re-
            gister das Ergebnis enthält".
```

Wir werden zunächst die beiden in NUM1 und NUM2 abgelegten Acht-Bit-Zahlen addieren, das Ergebnis in NUM1 speichern und den Überlauf ignorieren, wenn das Ergebnis von einer Acht-Bit-Zahl nicht dargestellt werden kann. Danach addieren wir das Ergebnis dieser beiden Speicherstellen nochmals, speichern das 16-Bit-Ergebnis aber in NUM3.

Erstes Beispiel:

```
LDA NUM1    erste Zahl nach A kopieren
ADDA NUM2   zweite Zahl addieren
STA NUM1    Ergebnis in NUM1 speichern
```

Zweites Beispiel:

```
LDB NUM1    erste Zahl nach B kopieren
SEX         die Acht-Bit-Zahl in B in eine
            16-Bit-Zahl in D umwandeln
STD NUM3    D nach NUM3 kopieren
LDB NUM2    die zweite Zahl nach B
            kopieren
SEX         in eine 16-Bit-Zahl in D
            wandeln
ADDD NUM3   die erste 16-Bit-Zahl von
            NUM3 nach D addieren
STD NUM3    das Ergebnis in NUM3
            speichern
```

Feldarbeit

In diesem Artikel untersuchen wir den Aufbau der Assemblersprache des 6809 im einzelnen und gehen dabei auf neue Assembleranweisungen und Adressierarten ein.

Ein Assemblerbefehl besteht aus drei Teilen (auch Felder genannt), die jedoch nicht in jeder Zeile enthalten sein müssen:

● **Das Labelfeld** befindet sich in der linken Bildschirmspalte. Labels sind Namen, die Zahlen darstellen und normalerweise Speicheradressen angeben. Sie können eine Länge von einem bis zu sechs Zeichen haben und müssen mit einem Buchstaben anfangen. Labels dürfen Registernamen, Assembler-Op-codes oder anderen, bereits definierten Labels nicht gleichen. Da diese Standardvereinbarungen jedoch nicht strikt festgelegt sind, können einzelne Assembler durchaus anderen Regeln folgen. Ein Leerzeichen am Zeilenanfang gibt an, daß das Labelfeld leer ist. Auch am Ende eines Labelfeldes steht ein Leerzeichen. So ist LABLDA ein gültiges Label, LAB LDA dagegen wird als Label LAB, gefolgt von dem Op-Code LDA, interpretiert.

Der Assembler unterhält einen Speicherzähler, der dem Befehlszähler des Prozessors ähnlich ist. Er enthält die Speicheradresse, an der das nächste Befehls- oder Datenbyte untergebracht werden soll. Trifft der Assembler auf ein Label, legt er dessen Namen in einem Speicherbereich ab, der Symboltabelle heißt und einem BASIC-Array ähnelt. Außer dem Namen wird dort auch die Adresse des Speicherzählers angegeben, die den Punkt bezeichnet, an dem der Assembler dieses Label erstmals gefunden hat. Wird nun der Assembler mit einem Label konfrontiert, sucht er zunächst in seiner Symboltabelle, ob dieses Label bereits existiert. Wenn er es findet, wird das Label gegen diese Adresse ausgetauscht, wenn nicht, speichert er es dort mit dem Inhalt des Speicherzählers.

● **Das Befehls- oder Op-Code-Feld** liegt neben dem Labelfeld. Es enthält ein mnemotisches Kürzel (normalerweise drei Buchstaben) und – falls nötig – einen Registernamen. So besteht ADDA aus dem Kürzel ADD und dem Registernamen A. Der Op-Code stellt den auszuführenden Prozessorvorgang dar. Wie das Labelfeld schließt auch er mit einem Leerzeichen ab.

● **Das Operanden- oder Adreßfeld** enthält Informationen über die Daten, mit denen der Op-Code arbeiten soll. Diese Daten sind entweder Adressen oder Labels, die wiederum Adressen darstellen.

Der Assemblerbefehl

```
LABEL1 ADDA NUM1
```

bedeutet „addiere die Daten, die in der durch das Symbol NUM1 dargestellten Adresse gespeichert sind, in den Akkumulator A". Die Adresse dieses Befehls ist nun als LABEL1 gespeichert. Wenn man auf diesen Befehl verzweigen will, genügt es, das Label als Sprungadresse anzugeben. NUM1 wurde ebenfalls als Label definiert und stellt eine Adresse dar, an der Daten gespeichert sind.

```
CLRA
```

ist ein Beispiel für einen Op-Code, der keinen Operanden benötigt. Er bedeutet „lösche den Akkumulator A – das heißt, setze seinen Inhalt auf Null". Beachten Sie, daß diese Zeile kein Label enthält. Zwar ist der Einsatz von Labels nicht zwingend vorgeschrieben, doch kann mit anderen Anweisungen leichter darauf verzweigt werden. Mit Labels lassen sich auch wichtige Befehle markieren und mit Bemerkungen versehen.

Derart eingesetzte Labels sind jedoch kein vollwertiger Ersatz für ausführliche Kommentare. Bemerkungen lassen sich in jede Zeile eintragen. Es muß nur nach dem letzten Zeichen des Operanden ein Leerzeichen eingegeben werden, dann kann der Kommentar aufgeführt werden. Manche Assembler benötigen Spezialzeichen, um den Anfang des Kommentarfeldes zu kennzeichnen. Ganze Kommentarzeilen werden normalerweise mit einem Stern eingeleitet.

Im Operandenfeld stehen auch die Konstanten in Form von Zahlen oder Zeichenketten. Zahlen werden dezimal interpretiert, wenn sie nicht anders gekennzeichnet sind: durch ein vorangehendes $ oder ein nachstehendes H (beispielsweise $AF08 oder AF08H) als Hexadezimalzahl, durch ein voranstehendes @ oder nachfolgendes Q (etwa @6712 oder 6712Q) als Oktalzahl (Zahlenbasis Acht) oder durch ein voranstehendes % oder nachfolgendes B (beispielsweise %11010011 oder 11010011B) als Binärzahl. Auch der ASCII-Code eines Zeichens kann eine Zahl angeben, wenn ihm ein Apostroph voransteht – 'A bedeutet dann entweder 65 oder $41.

Vielseitig einsetzbar ist auch der aktuelle Wert des Speicherzählers. Zwar ist dieser Wert beim Aufruf des Programms zumeist nicht bekannt, doch ist es möglich, sich im Operandenfeld mit einem Stern darauf zu beziehen. Die meisten Assembler können diese Angabe

auch in einfachen mathematischen Ausdrükken verarbeiten.

```
LDA *+5
```

bedeutet beispielsweise „lade den Akkumulator mit der Speicherstelle, deren Adresse fünf Bytes über dem augenblicklichen Inhalt des Speicherzählers liegt".

Ein Assembler kann normalerweise eine Reihe von Anweisungen oder Pseudobefehlen verstehen, die wie normale Op-Codes in das Programm eingebaut sind. Zwei dieser Befehle haben wir schon früher kennengelernt:

```
CR FCB 13
```

FCB (Fix Constant Byte) reserviert an der aktuellen Speicheradresse ein einzelnes Byte, dem der Wert des Operanden zugewiesen wird. Der Befehl initialisiert eine Speicherstelle (durch das CR-Symbol bezeichnet) mit dem Wert 13.

```
MEMTOP FDB $7FFF
```

FDB (Fix Double Byte) führt den gleichen Vorgang für einen Zwei-Byte-Wert (16 Bits) aus. Speicherplatz kann aber auch ohne Angabe des Inhalts mit dem Pseudobefehl RMB (Reserve Memory Bytes) reserviert werden:

```
TAFEL1 RMB 7
TAFEL2 FCB  $F6
```

Die erste Zeile reserviert für eine Wertetabelle sieben Bytes, deren erste Adresse von dem Label TAFEL1 dargestellt wird. Wenn TAFEL1 daher zum Beispiel die Adresse $C104 kennzeichnet, dann repräsentiert TAFEL2 die um sieben Bytes höhere Adresse $C10B.

Auch ganze Zeichenfolgen lassen sich im Speicher ablegen:

```
ERRMSG FCC 'FEHLER
```

initialisiert sechs Speicherstellen mit den ASCII-Codes für F,E,H,L,E und R. Damit lassen sich auch in die Assemblersprache leicht Meldungen und Promptzeilen einfügen.

Ein weiterer wichtiger Pseudobefehl ist ORG (ORiGin), der den Speicherzähler auf den angegebenen Wert setzt. Er steht am Anfang eines Programmblocks und teilt dem Assembler mit, in welchen Speicherbereich er den Block beim Übersetzen in den Maschinencode ablegen soll. Selbst wenn Assembler eine Standard-Anfangsadresse haben, sollten Sie Ihre Programme immer mit einer ORG-Anweisung beginnen. Ein Programm kann mehrere ORG-Anweisungen enthalten. ORG wird ohne Label oder Operanden eingesetzt.

Die einzige Anweisung, die normalerweise vor dem Pseudobefehl ORG auftaucht, ist EQU (EQUate), die den Variablensymbolen Werte zuordnet. EQU bezieht sich nicht auf den Speicherzähler.

```
RESET EQU $F100
```

definiert das RESET-Symbol und weist ihm den Wert $F100 zu. RESET ist daher die Abkürzung einer Zahl, während ein Label am Anfang einer Befehlszeile immer eine Adresse darstellt, an der Daten oder Code gespeichert sind.

END teilt dem Assembler mit, wo sich das Programmende befindet. Dieser Pseudobefehl nimmt keine Label oder Operanden an.

Ein Maßstab für die Flexibilität von Assemblersprachen sind die Adressierarten – das heißt die Möglichkeiten, Operanden zu interpretieren. Alle bisher aufgeführten Befehle setzten entweder die ganze oder die erweiterte Adressierart ein. Dabei gibt der Wert oder das Label des Operandenfeldes die Speicheradresse mit den Daten an.

Ein-Byte-Adressen

Die direkte Adressierung hat als Operand nur eine Ein-Byte-Adresse, die der Prozessor als niederwertiges Byte einer vollständigen Zwei-Byte-Adresse ansieht. Das höherwertige Byte wird dem Kurzadressierregister entnommen, einem Acht-Bit-CPU-Register, das sich vom Programm ansprechen läßt. Diese Adressierart ist sehr flexibel und läßt sich universell einsetzen: Mit dieser Methode angelegte Subroutinen beziehen sich nicht auf einen festen Speicherbereich. Erst das Laden des Kurzadressierregisters vor Aufruf der Routinen gibt an, in welchem Speicherbereich Daten liegen.

Die erweiterte Adressierung benötigt als Operand eine Zwei-Byte-Adresse. Dieser Befehl bezieht sich immer auf ein festes Speicherbyte und ist daher sehr unflexibel. Der Assembler erkennt die direkte oder erweiterte Adressierung durch die Art des Operanden.

Oft eingesetzt wird auch die unmittelbare Adressierung, bei der die eigentlichen Daten in dem Operandenfeld selbst enthalten sind. Dabei steht vor dem Operanden ein #.

Beachten Sie, daß das Label NUM1 die Adresse $1000 darstellt. Sie mögen erwartet haben, daß die ORG-Anweisung auf Adresse $1000 liegt und der darauf folgende Befehl (und damit das Label NUM1) eine höhere

ORG $1000	Anfangsadresse des Maschinencodes ist $1000
NUM1 FCB $FFFF	initialisiert die Speicherstelle NUM1 mit $FFFF
LDD NUM1	lädt $FFFF, den Inhalt von NUM1 in das D-Register
LDD #NUM1	lädt $1000, den unmittelbaren Wert von NUM1, in das D-Register

Adresse hat. Bedenken Sie jedoch, daß ORG nur das Assemblerprogramm anweist, wie es die Übersetzung in den Maschinencode durchführen soll, und nicht Teil des Programms ist. ORG belegt also keinen Speicherplatz. NUM1 nimmt den Wert $1000 an, da der Assembler bei diesem Wert des Speicherzählers zum ersten Mal auf NUM1 trifft. Beachten Sie ebenfalls, daß LDD NUM1 den Inhalt der Speicherstelle NUM1 ($FFFF) in den Akkumulator D lädt.

Einfaches Rechnen

Nachdem in der letzten Folge der Aufbau des 6809 erklärt wurde, sehen wir uns nun an, wie sich mit seinem Befehlssatz einfache mathematische Aufgaben ausführen lassen. Besonders interessant sind dabei die Vorzeichen-Arithmetik und der Einsatz des Condition Code Registers in den Programmen.

In dieser Folge setzen wir erstmals Befehle zu einem Programm zusammen. Dazu führen wir zunächst einige neue Kommandos ein und sehen uns die Darstellung von Daten genauer an. Unser Beispielprogramm wandelt eine „Binär Codierte Dezimalzahl" (BCD) in das Binärformat um.

Dezimalzahlen im BCD-Format eignen sich ausgezeichnet für die Acht-Bit-Verarbeitung. Die einzelnen Ziffern der Dezimalzahlen werden dabei als vierstellige Binärzahlen geschrieben. So wird aus der Dezimalzahl 69 die BCD-Zahl %01101001. Die vier Bits auf der linken Seite (0110) sind die binäre Entsprechung der Zahl 6, während die vier rechten Bits (1001) die 9 darstellen. Das BCD-Format ergibt einen völlig anderen Dezimalwert als die Umwandlung einer Binärzahl (%01101001 ergibt dezimal 105).

Für unser Umwandlungsprogramm benötigen wir eine Reihe neuer Befehle:

● LSR (Logische Rechtsverschiebung): Alle Bits des Operanden werden um eine Position nach rechts geschoben. Das rechtsaußen stehende Bit wandert in das Übertragsbit des Condition Code Registers, und das linksaußen stehende Bit wird auf Null gesetzt.

● AND: Für jedes Bit eines Registers wird eine logische Verknüpfung mit den entsprechenden Bits des Operanden durchgeführt und das Ergebnis wieder im Register gespeichert. Mit diesem Vorgang lassen sich Bits „maskieren". Steht ein Bit des ersten Registers auf Eins, dann kopiert AND den Bitwert des zweiten Registers in das Ergebnisregister. Steht das Bit des ersten Registers dagegen auf Null, dann ist unabhängig vom Inhalt des zweiten Registers auch das Ergebnisbit Null. Eine logische AND-Verknüpfung mit %00001111 kopiert daher nur die vier rechten Bits in das Ergebnisregister:

```
%00001111 Registerwert
%10110110 AND-Wert des Operanden
%00000110 Ergebnis im Register
```

● MUL: Multipliziert den Inhalt der Register A und B und speichert das Ergebnis im D-Register (das aus A und B zusammengesetzte 16-Bit-Register). Nur wenige Acht-Bit-Prozessoren unterstützen einen Op-Code für Multiplikation.

● SWI (SoftWare Interrupt): Dieser Befehl bietet eine Möglichkeit, Maschinencodeprogramme zu beenden und die Steuerung an das Betriebssystem zurückzugeben. Wir werden diesen Befehl genauer behandeln, wenn wir auf das Interruptsystem eingehen. Hier das Programm für die Umwandlung BCD in Binär:

● Der Anfangswert des Speicherzählers:

```
ORG $1000
```

● BCD 58 in BCDNUM speichern und ein Byte in BINNUM reservieren:

```
BCDNUM FCB  %01011000
BINNUM  RMB 1
```

● BCD 58 in das A-Register laden, die niederwertige Ziffer maskieren und anschließend in BINNUM speichern:

```
START LDA   BCDNUM
      ANDA #%00001111
      STA   BINNUM
```

● BCD 58 in Akkumulator A laden und die vier hohen (linksstehenden) Ziffern nach rechts schieben:

```
SHIFT LSRA
      LSRA
      LSRA
      LSRA
```

● Die Dezimalzahl 10 in das B-Register laden und mit dem Inhalt von A multiplizieren:

```
MULT LDB  #10
     MUL
```

● Ergebnis sind die 16 Bits des D-Registers. Da das Resultat nicht über 90 liegen kann (10 * 9 = 90), wird nur das niederwertige Byte von D gebraucht. Sein Inhalt wird auf BINNUM addiert und gespeichert.

```
ADDIT ADDB BINNUM
      STB   BINNUM
```

● Die BCD-Zahl befindet sich nun in BCDNUM und ihre binäre Entsprechung in BINNUM. Wir brauchen nur noch auf Betriebssystemebene zurückzukehren und den Quellencode zu beenden:

```
RETURN SWI
END
```

Es gibt mehrere Möglichkeiten, positive und negative Zahlen darzustellen. Für den Arbeits-

speicher und die Register eines Computers ist das **Zweierkomplement** die einfachste Methode, mit negativen Zahlen umzugehen. Das Zweierkomplement einer Binärzahl entsteht, wenn alle Ziffern in ihr Gegenteil verkehrt (alle Nullen in Einsen und umgekehrt) und die Zahl Eins addiert wird. So ist das Zweierkomplement von 0101 die Binärzahl 1011.

Wie läßt sich diese Zahl nun für die Vorzeichen-Mathematik verwenden? Sehen wir uns zunächst an, welcher Zahlenbereich überhaupt möglich ist. Ein Acht-Bit-Register hat 256 verschiedene Bitmuster, die positive Zahlen von 0 bis 255 **oder** negative und positive Zahlen von −128 bis +127 darstellen können. Die Tabelle zeigt die Binärdarstellung von −7 bis +7 im Vier-Bit-Format.

Sie werden bemerken, daß alle negativen Zahlen der Tabelle als höchstwertiges Bit (linksaußen) eine Eins haben und alle positiven Zahlen eine Null.

Aus dieser Vier-Bit-Tabelle lassen sich nun einige grundlegende Eigenschaften der Vorzeichen-Mathematik mit dem Zweierkomplement ableiten:

- Aus dem Zweierkomplement einer negativen Zahl ergibt sich das positive Äquivalent (und umgekehrt).
- Das höchstwertige Bit ist bei positiven Zahlen immer Null und bei negativen Zahlen immer Eins. Damit lassen sich positive und negative Zahlen leicht auseinanderhalten.
- Das Zweierkomplement von Null ist Null (1111 plus 1).
- Addition und Subtraktion werden wie üblich ausgeführt, wobei das Ergebnis immer das richtige Vorzeichen erhält.

Steuermechanismen

Überprüfen Sie die letzte Aussage einmal mit einigen einfachen Additionen und Subtraktionen. Die Multiplikation von Zahlen mit Vorzeichen ist komplizierter, da der Befehl MUL, den wir schon bei der Umwandlung BCD-in-Binär eingesetzt hatten, den Inhalt der Register A und B nur als Zahlen ohne Vorzeichen behandelt. Die Multiplikation zweier Zahlen mit Vorzeichen muß extra programmiert werden.

Die engen Grenzen unseres bisherigen „linearen" Programmaufbaus können wir nur mit folgenden Steuermechanismen erweitern:

- Auswahl: die Wahl zwischen zwei unterschiedlichen Lösungswegen (wie der IF-Befehl in BASIC).
- Wiederholung: der wiederholte Ablauf einer Befehlsfolge:

1) wenn eine bestimmte Bedingung „wahr" bleibt (die WHILE. . .WEND-Struktur);
2) bis eine bestimmte Bedingung „wahr" wird (REPEAT. . .UNTIL); oder
3) für eine festgelegte Anzahl von Durchläufen (FOR. . .NEXT).

All diese Strukturen sind nur möglich, wenn wir auch testen können, ob eine Bedingung wahr oder falsch ist. Dabei wird zumeist überprüft, ob eine Variable einen bestimmten Wert hat oder nicht. Doch obwohl die Assemblersprache leicht zwei Registerwerte miteinander vergleichen kann, würden damit im Normalfall nur zwei Möglichkeiten zur Verfügung stehen: Ein Wert ist Null oder nicht Null, und ein Wert ist positiv oder negativ. Mit Zusatzbefehlen lassen sich jedoch auch andere Arten von Tests durchführen.

Viele Tests werden erst durch den Einsatz des Condition Code (CC) Registers möglich. Das CC-Register hat ein Acht-Bit-Format, bei dem uns jedoch nicht der dort gespeicherte Wert interessiert, sondern der Status (1 oder 0) jedes einzelnen der acht Bits. Fünf der acht Bits zeigen Bedingungen an, die wir für die oben erwähnten Verzweigungen einsetzen können, die anderen drei beziehen sich auf

Die folgenden Beispiele zeigen, wie sich Assembleranweisungen und Befehle auf den Speicherzähler des Assemblers und die Speicherstellen auswirken.

Assembleranweisungen

Label-Feld	Op-Code-Feld	Operand-Feld	Speicherzähler	Speicherinhalt	
* ——— BEISPIEL ———					
RESET	EQU	$F100	$0400	???	Da kein ORG angegeben ist, wird die Speicheradresse auf die Standardanfangsadresse des Assemblers gesetzt. EQU wird nicht beeinflußt.
INDEX	EQU	16	$0400	???	
MASK1	EQU	%01101010	$0400	???	
	ORG	$1000	$1000	???	Setzt die Anfangsadresse, wie von ORG angegeben.
CR	FCB	16	$1000	$10	FCB speichert den Operanden in dem Byte, das der Speicherzähler angibt.
MEMTOP	FDB	$7FFF	$1001	$7F	Zwei Bytes werden mit FDB initialisiert.
			$1002	$FF	
TABLE1	RMB	7	$1003	???	RMB reserviert sieben Bytes (mit undefiniertem Inhalt), indem er den Speicherzähler um sieben inkrementiert.
			$1004	???	
			$1005	???	
			$1006	???	
			$1007	???	
			$1008	???	
			$1009	???	
ERRMSG	FCC	'ERROR	$100A	$45	Der ASCII-Code des Operanden-Strings wird mit der FCC-Anweisung im Speicher abgelegt.
			$100B	$52	
			$100C	$52	
			$100D	$4F	
			$100E	$52	
	CLRA		$100F	$4F	Schließlich ein Assemblerbefehl. Da er keinen Operanden hat, besteht der Op-Code nur aus einem Byte.
	END		$100F	???	Eine Anweisung, die den Speicherzähler nicht beeinflußt.

* ——— SYMBOLTABELLE ———

RESET	F100	INDEX	0010	MASK1	006A
CR	1000	MEMTOP	1001	TABLE1	1003
ERRMSG	100A				

So sind die Symbole in dem Arbeitsspeicher des Assemblers untergebracht.

Dezimal	Binär
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

die Behandlung von Interrupts, auf die wir später genauer eingehen. Eins der fünf Bits – das interne Übertragsflag H – wird nur für die BCD-Arithmetik gebraucht und ist für uns daher im Augenblick nicht wichtig. Die anderen Flags haben folgende Aufgaben:

- C: Das Übertragsbit zeigt an, ob bei einer Addition oder Subtraktion ein Übertrag aufgetreten ist. Bei Verschiebungen des Akkumulatorinhalts kann es außerdem das „herausgefallene" Bit speichern. Für die Feststellung, ob eine Zahl gerade oder ungerade ist, wird beispielsweise das niederwertigste Bit nach C verschoben und dort getestet. Flag C ist Bit 0 (das niederwertigste Bit) des CC.
- V: Das Überlaufbit wird auf Eins gesetzt, wenn das Ergebnis eines arithmetischen Vorgangs zu groß für das Ergebnisregister ist. Flag V ist Bit 1 des CC.
- Z: Das Null-Flag wird auf Eins gesetzt, wenn der Inhalt eines Registers Null ist. Flag Z ist Bit 2 des CC.
- N: Das Negativ-Flag ist eine Kopie des höchstwertigen (Vorzeichen-)Bits eines Registers. Steht das Flag auf Eins, dann ist die Zahl negativ. Flag N ist Bit 3 des CC.

Verzweigungsbefehle

Eine der schwierigsten Aufgaben der Assemblersprache ist der richtige Einsatz der Status-Flags. Nicht jeder Befehl ändert die Flags. Einige Flags werden nur in Abhängigkeit vom Akkumulatorinhalt gesetzt, andere dagegen auch von anderen Registern. Am einfachsten ist es, nur den Speicherinhalt des Akkumulators zu testen und den Test an einem Punkt auszuführen, an dem die verschiedenen Flags noch nicht von dazwischenliegenden Befehlen verändert wurden.

Flags lassen sich mit Verzweigungsbefehlen testen, die eine maschinennahe Version des BASIC-Befehls GOTO sind. Der 6809 setzt fast nur relative (und kaum absolute) Verzweigungsbefehle ein. **Relative Verzweigungen** überspringen von der aktuellen Befehlsadresse an vorwärts oder rückwärts eine programmierte Anzahl Bytes, **absolute Verzweigungen** dagegen verzweigen direkt zu einer angegebenen Adresse. Beide Befehlsarten haben jedoch die gleiche Wirkung. Es gibt kurze Verzweigungen, deren Bereich in einem Byte angegeben wird (−128 bis 127), und sogenannte **Weitverzweigungen,** denen jede Adresse des Speichers zugänglich ist. Wir werden nur kurze Verzweigungen verwenden.

Der 6809 besitzt viele Verzweigungsbefehle, die wir nach und nach vorstellen werden. Die folgenden Beispiele zeigen, wie die Verzweigungsbefehle durch das Testen und Vergleichen von Akkumulatorwerten Auswahl und Wiederholung ermöglichen.

- ANDCC: Vor Einsatz des Condition Code Registers sollten routinemäßig alle Flags auf Null gesetzt werden. Zwar kann dieses Register nicht direkt mit Werten geladen werden, doch erzielt der ANDCC-Befehl die gleiche Wirkung. Wie bei dem AND-Befehl werden die Bits mit Nullen maskiert.
- SUB (SUBtrahieren): Vom Akkumulatorinhalt wird der Operand abgezogen und je nach Ergebnis das Flag C, V, Z und N gesetzt. (Bei Acht-Bit-Subtraktionen wird ebenfalls das H-Flag gesetzt.)
- CMP: Dieser Befehl arbeitet wie SUB, verändert aber nicht den Registerinhalt. Wie bei SUB werden die Flags C, V, Z und N (möglicherweise auch H) gesetzt.
- BRA (unbedingte Verzweigung): entspricht dem BASIC-Befehl GOTO.
- BGT (verzweige, falls größer als Null-Vorzeichentest): Die Verzweigung wird ausgeführt, wenn Z gleich Null ist (der Registerinhalt ist ungleich Null) und N und V entweder beide auf Null oder beide auf Eins stehen. Diese Bedingung ist erfüllt, wenn nach einer Subtraktion oder einem Vergleich der Registerinhalt größer ist als der Speicheroperand. Ähnliche Tests für Zahlen mit Vorzeichen sind BGE, BLT und BLE.
- BLO (verzweige, falls niedriger als Null): Dieser Test prüft nicht auf Vorzeichen, da es sinnlos wäre, bei Zahlen ohne Vorzeichen Flag N abzufragen. Die Verzweigung tritt ein, wenn das C-Flag gesetzt ist und damit ein negativer Übertrag nach einer Subtraktion angezeigt wird. BLS, BHI und BHS arbeiten auf ähnliche Weise.
- Das folgende Programm soll die größere von zwei Acht-Bit-Zahlen mit Vorzeichen finden. Die Zahlen sind in $3000 und $3001 gespeichert. Zunächst werden die Zahlen mit Labeln versehen:

```
NUM1 EQU $3000
NUM2 EQU $3001
ANS  EQU $3002
     ORG $1000
```

- Am Anfang des Programms werden die Flags des Condition Code Registers auf Null gesetzt, die erste Zahl geladen und mit der zweiten verglichen:

```
ANDCC #%11110000
LDA   NUM1
CMPA  NUM2
```

- Wenn NUM1 größer ist, verzweigt das Programm auf FINISH. Ist dies nicht der Fall, wird die zweite Zahl in das A-Register geladen. Die Zahl, die sich beim Sprung auf FINISH in diesem Register befindet, wird dann in ANS gespeichert. Das Programm kehrt zum Betriebssystem zurück und ENDet:

```
      BGT FINISH
      LDA NUM2
FINISH STA ANS
      SWI
      END
```

Adressenänderung

Wir untersuchen, wie die beiden Indexregister X und Y für die indizierte Adressierung eingesetzt werden, und zeigen an einigen Beispielen die Vorteile dieser Methode.

Das ursprüngliche Konzept des programmierbaren Computers basierte darauf, daß das Programm in dem gleichen Bereich (und im gleichen Format) wie seine Daten untergebracht sein sollte, damit es sich während des Ablaufs selbst modifizieren kann. Diese Möglichkeit war jedoch nicht für eine Umstellung der Befehle gedacht, sondern es sollten hauptsächlich Adressen geändert werden, von denen die Programme Daten abriefen. Stellen Sie sich nun vor, Sie sollten eine Tabelle mit mehreren tausend Zahlen bearbeiten, hätten dafür aber nur Befehle zur Verfügung, die, sehr zeitaufwendig, jeweils eine einzige Adresse ansprechen können.

Durch veränderbare Adressen ließe sich das Problem leicht lösen. Dabei würden in einer Schleife Befehle wiederholt werden, die sich bei jedem Durchlauf auf eine andere Adresse beziehen. Die Adreßänderung wird durch die Veränderung eines Registerwertes ausgelöst, der die Adresse beeinflußt. Dieses Konzept erscheint oft in BASIC-Programmen:

```
FOR I = 1 TO N
PRINT TABELLE(I)
NEXT I
```

In diesem Modul bezieht sich der PRINT-Befehl auf immer andere Daten, da sich das grundlegende Datenobjekt (TABELLE) über den ansteigenden Indexwert (I) bei jedem Durchgang verändert.

Das Grundprinzip der „Indizierten Adressierung" besteht darin, den Inhalt des Indexregisters auf eine Basisadresse zu addieren, um die aktuelle Adresse zu erhalten – das heißt die Speicherstelle, die angesprochen werden soll. Tritt dieser Befehl im Inneren einer Schleife auf, läßt sich dort auch das Indexregister inkrementieren oder dekrementieren, so daß die Tabellenwerte nacheinander abgerufen werden.

Der 6809 hat für diese Aufgabe nicht nur die Register X und Y, sondern auch S und U zur Verfügung. Unter besonderen Umständen läßt sich auch der Befehlszähler einsetzen. Weiterhin gibt es eine ganze Reihe von Indizierungsmöglichkeiten, die fast alle Programmanforderungen erfüllen. Da wir von nun an die verschiedenen Formate der Indizierung einsetzen werden, haben Sie ausreichend Gelegenheit, sich damit vertraut zu machen.

Die indizierte Adressierung wird durch den Zusatz ,X im Operandenfeld angezeigt (natürlich nur, wenn X das eingesetzte Indexregister ist). Indizierte Befehle haben folgendes Standardformat:

```
Opcode    Offset, Indexregister
  LDA     TABELLE1,X
  STA     TABELLE2,Y
```

Ein Offset von Null kann weggelassen werden:

```
Opcode    ,Indexregister
  LDA     ,X
  STA     ,Y
```

Sehen wir uns diesen Vorgang einmal in der Praxis an. Angenommen, von $3000 an ist eine Tabelle mit 64 Acht-Bit-Werten gespeichert, die wir nacheinander adressieren wollen. Dazu definieren wir die Basisadresse und reservieren mit Assembleranweisungen den Platz für die Tabelle:

```
          ORG   $3000
TABELLE   RMB   64
```

Diese Anweisungen setzen den Befehlszähler auf $3000, definieren den Anfang von TABELLE als $3000 und reservieren die nächsten 64 Bytes. Mit dem nächsten Programmteil adressieren wir die Bytes: Die neue ORG-Anweisung zeigt an, daß der Code in einem anderen Teil des Speichers abgelegt ist als die Daten. Mit dieser Vorsichtsmaßnahme verhindern wir, daß die Daten unbeabsichtigt den Programmcode überschreiben.

```
          ORG   $1000
COUNT     FCB   0
          LDX   #0
LOOP      LDA   TABELLE,X
```

Wir verändern nun den Wert des X-Registers:

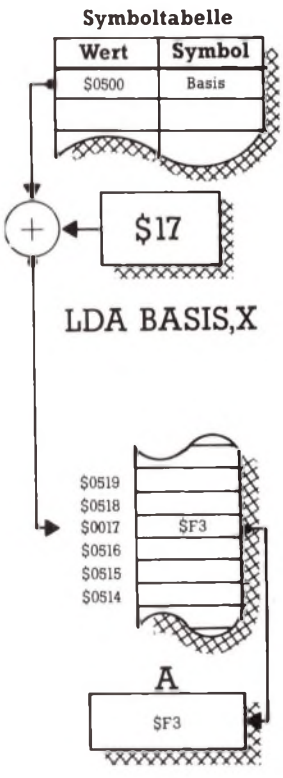
```
          TFR   X,D
          ADDD  #1
          TFR   D,X
```

Diese etwas umständliche Methode, X zu inkrementieren, läßt sich gut für In- und Dekremente einsetzen, die über Zwei liegen. Weitere Möglichkeiten sehen wir uns später an. Das letzte Teil des Codes inkrementiert den Zähler (COUNT) und stellt sicher, daß er nicht auf 64 steht (in diesem Fall ist das Programm beendet – die Schleife muß nicht nochmals durchlaufen werden):

```
          INC   COUNT
          LDB   COUNT
          CMPB  #64
          BLT   LOOP
```

Der Code läßt sich natürlich noch verbessern. Am praktischsten ist die Methode der „automatischen Inkrementierung". Der Befehl:

Das Symbol BASIS – mit $0500 initialisiert – ist die Adresse des ersten Bytes einer Wertetabelle. Der Befehl LDA BASIS,X setzt die Methode der indizierten Adressierung ein. Dabei wird der Inhalt des X-Registers auf den Wert von BASIS addiert und so die aktuelle Adresse errechnet, deren Inhalt in den Akkumulator geladen wird. Wenn sich dieser Befehl innerhalb einer Schleife mit dem Zähler X befindet, läßt sich die gesamte Tabelle Byte für Byte ansprechen. Da X ein 16-Bit-Register ist, kann die Schleife (in einem Acht-Bit-System wie dem 6809) den gesamten Speicherbereich ($0000 bis $FFFF) adressieren.



```
        LDA   TABELLE,X+
```

veranlaßt, daß der Wert von X automatisch nach jedem Gebrauch inkrementiert wird. Für eine Tabelle von 16-Bit-Werten wird X mit

```
        LDA   TABELLE,X++
```

um jeweils zwei inkrementiert. Unser ursprüngliches Programm nimmt nun Formen an:

```
LOOP    LDA   TABELLE,X+
        INC   COUNT
        LDB   COUNT
        CMPB  #64
        BLT   LOOP
```

Die Tabelle läßt sich mit der automatischen Dekrementierung auch von hinten nach vorn ansprechen. Diese Methode hat den Vorteil, daß der letzte Wert des X-Registers Null ist. Da die automatische Dekrementierung eines Indexregisters auch die Flags des Condition Code Registers setzt, können wir das Ende der Schleife direkt abfragen, ohne den Befehl CMP einsetzen zu müssen. Die gleiche Wirkung erzielt auch das Laden des Indexregisters mit einem negativen Wert, der so lange inkrementiert wird, bis Null erreicht ist. Bei jeder automatischen Inkrementierung werden die Flags des Condition Code Registers gesetzt, um das jeweilige Resultat anzuzeigen. Das Ergebnis Null setzt beispielsweise das Null-Flag, ein Übertrag das Übertrags-Flag und so weiter.

Vorsicht Bearbeitungsmodule!

Tests sollten möglichst nur über den Akkumulator durchgeführt werden, da in den meisten Programmen zwischen dem In- oder Dekrementierbefehl und dem Testbefehl Bearbeitungsmodule liegen, die das Condition Code Register verändern.

Selbst wenn Sie die Tabelle nicht von hinten nach vorn abfragen, sollte der Zähler rückwärts laufen, damit die Schleife bei Null endet. Bei der automatischen Dekrementierung ist wichtig, daß die Dekrementierung vor der Adressenberechnung ausgeführt wird, während die automatische Inkrementierung das Register nach der Berechnung der Adresse erhöht. Wenn X dabei 7 enthält und die TABELLE bei $1000 beginnt, dann lädt der Befehl LDA TABELLE,X+ erst den Inhalt der Adresse $1007 in den Akkumulator und inkrementiert danach X von 7 auf 8. LDA TABELLE,−X (beachten Sie, daß das Minuszeichen hier vor dem Registernamen steht) dekrementiert zunächst X von 7 auf 6 und lädt dann den Akkumulator mit dem Wert der Adresse $1006.

Wenn wir rückwärts durch die Tabelle gehen und der Zähler im Register B liegen soll, muß die Schleife folgendermaßen aussehen:

```
        LDX   #64
        LDB   #64
LOOP    LDA   TABELLE,-X
        DECB
        BGE   LOOP
```

In unserem ersten Beispielprogramm zählt eine einfache Schleife die Anzahl der negativen Werte einer Tabelle von Acht-Bit-Werten. Der Zähler im Akkumulator B dient dabei auch als Offset des festen Wertes im X-Register. Das zweite Programm zeigt, wie beide Indexregister zusammen und mit einem Null-Offset eingesetzt werden. Das Programm kopiert einen String von einer Speicherstelle (möglicherweise einem Eingabebuffer) in einen anderen Speicherbereich. Der String schließt mit einem Return-Zeichen ab. Seine Länge ist zwar nicht bekannt, kann jedoch maximal 255 Zeichen betragen. Nach der Speicherung wird das Return-Zeichen gelöscht und ein Byte, das die Länge des Strings anzeigt, an seinen Anfang gesetzt.

Im ersten Programm ist eine Tabelle von Acht-Bit-Werten von der Speicherstelle $3001 an untergebracht. Die Anzahl der Werte ist in $3000 gespeichert. Sie liegt unter 256. Das Programm zählt die Anzahl der negativen Werte in der Tabelle. Das zweite Programm kopiert einen String vom Eingabebuffer bei $3000 in den nächsten freien Stringbereich, dessen Adresse in $0010 liegt.

Zähler für negative Werte

```
TABELLE EQU  $3000      TABELLE enthält die Anzahl der Werte
        ORG  $1000
NEGS    FCS  0         NEGS soll die Anzahl der negativen Werte
        LDB  TABELLE    enthalten
        LDX  #TABELLE  Die Adresse der Tabelle wird in X
                        gespeichert. Den letzten Wert der Tabelle
LOOP    LDA  B,X        holen
        BGE  ENDLP     GOTO ENDLP, wenn der Wert positiv ist
        INC  NEGS      ELSE erhöhe den Zähler, wenn der Wert
                        negativ ist
ENDLP   DECB           Tabelle durchsuchen
        BGT  LOOP      Gibt es weitere Werte? Falls ja, GOTO LOOP
        SWI            ELSE zurück zum Betriebssystem
        END
```

Kopiermodul für Strings

```
CR      EQU  13        ASCII-Wert des Zeichens RETURN
STRING1 EQU  $3000     Adresse des Eingabebuffers
STRING2 EQU  $0010     STRING2 enthält die Adresse des freien
        ORG  $1000      Speicherbereichs
        LDX  #STRING1  Die Adresse des Originalstrings wird in X
                        geladen
        LDY  STRING2   Die Adresse des Ziel-Strings (die in $10,$11
                        gespeicherte Adresse) wird in Y geladen
        TFR  Y,U       Die Adresse des Längen-Bytes in U
                        übertragen
        LDA  #0        Null im ersten Byte des Ziel-Strings
        STA  ,Y+        speichern
LOOP    LDA  ,X+       Das nächste Zeichen aus dem Eingabebuffer
                        holen
        CMPA #CR       Ist es das RETURN-Zeichen?
        BEQ  FINISH    Wenn ja, dann Ende
        STA  ,Y+       ELSE kopieren (und) Eins zur Länge
        INC  ,U         addieren
        BRA  LOOP      Nächstes Zeichen
FINISH  SWI            Zum Betriebssystem zurückkehren
END
```

String-Vergleiche

Wir untersuchen, wie mit den Indexregistern einfache mathematische Operationen durchgeführt werden und wie Unterroutinen funktionieren. Außerdem wird ein Programm entwickelt, das Strings miteinander vergleicht.

In der vorigen Folge erhielten Sie einen ersten Einblick in die Arbeitsweise der indizierten Adressierung des 6809. Dabei wird die aktuelle Adresse – beispielsweise OFFSET,X – aus dem Offset (Konstante oder Speicherstelleninhalt) und dem gegenwärtig im Indexregister gespeicherten Wert errechnet. Ein Offset von Null kann mit ,X angegeben werden (0,X ist ebenfalls richtig), und auch die Akkumulatoren A, B oder D lassen sich für diese Aufgabe einsetzen (zum Beispiel B,X). Beim Abarbeiten von Tabellen kann die automatische In- und Dekrementierung viel Arbeit ersparen. Dabei wird das Register entweder nach Ausführung eines Befehls um Eins oder Zwei erhöht (,X+ und ,X++) oder zuvor um Eins oder Zwei vermindert (,−Y und ,−−Y).

Der Befehl LEA (Lade Adresse) kann bei der indizierten Adressierung einfache mathematische Operationen mit den Werten des Indexregisters ausführen. Da normale mathematische Befehle nur mit Akkumulatorwerten und nicht mit anderen Registern arbeiten, läßt sich zwar der Inhalt des Indexregisters in den Akkumulator D übertragen, dort in- oder dekrementieren und wieder zurückladen, doch dieser Vorgang ist sehr langsam und umständlich. LEA arbeitet zwar nur mit den Registern X, Y, S und U, führt aber alle notwendigen Adreßberechnungen durch und lädt dann die aktuelle Adresse. Da andere Ladebefehle normalerweise den Inhalt der aktuellen Adresse laden, ist dies eine brauchbare Alternative.

Sehen wir uns ein Beispiel an:

```
LEAX  −1,X
```

berechnet die aktuelle Adresse aus der Summe von −1 und dem augenblicklichen Inhalt des X-Registers. Diese Adresse wird nach X zurückgeladen und dekrementiert so den in diesem Register gespeicherten Wert. Der Befehl hat aber noch andere Funktionen: Damit Adressen nicht mehrfach berechnet werden müssen, kann er die Berechnung beispielsweise einmal ausführen und das Ergebnis speichern.

Mit dem Befehl ABX (Addiere den Inhalt des Akkumulators B auf das X-Register) kann auch das X-Register in Berechnungen einbezogen werden. Diese Anwendung ist jedoch nicht so universell einsetzbar wie LEA.

Subroutinen sind eigenständige Programmodule, die vom Hauptprogramm (oder anderen Subroutinen) aus aufgerufen werden und spezielle Aufgaben ausführen. Nach deren Beendigung übernimmt das aufrufende Modul wieder die Steuerung und führt nun den Befehl aus, der unmittelbar hinter dem Unterprogrammaufruf steht. Subroutinen werden hauptsächlich aus drei Gründen eingesetzt:

1) Der Code muß nur einmal geschrieben werden. Es ist praktisch, oft eingesetzte Funktionen als Subroutinen anzulegen, die bei Bedarf aufgerufen werden.

2) Es entsteht eine Sammlung universell verwendbarer Routinen, die sich in unterschiedlichen Programmen einsetzen lassen.

3) Programme, die in kleine Module aufgeteilt sind, lassen sich leichter handhaben.

Bei Unterprogrammen der Assemblersprache muß unbedingt beachtet werden, daß das aufrufende Programm und die Subroutine die gleichen Register einsetzen. In der Maschinencodeprogrammierung treten häufig Fehler auf, wenn nach Speicherung eines Registerwertes eine Subroutine aufgerufen wird, die diesen Wert verändert. Es sollten deshalb nicht nur alle von der Subroutine eingesetzten Register bekannt sein und dokumentiert werden, sondern auch der Registerinhalt sollte möglichst vor Aufruf der Routine gespeichert und nach Beendigung wieder geladen werden.

BSR- und JSR-Befehle

Wir werden uns später ansehen, wie der Stack (Stapelspeicher) diese Aufgabe ausführt und Werte an die Subroutinen übergeben kann. Für den Augenblick gehen wir jedoch davon aus, daß die Subroutine die gleichen Daten (globale Variablen) einsetzt wie das aufrufende Programm und daß alle weiteren Werte in den Registern abgelegt sind. Subroutinen werden mit folgenden Befehlen aufgerufen:

- BSR: Verzweige zum Unterprogramm
- JSR: Springe zum Unterprogramm

Der BSR-Befehl veranlaßt eine relative Verzweigung. Das heißt, er spricht eine Subroutine an, die eine bestimmte Anzahl Schritte vom augenblicklichen Wert des Befehlszählers entfernt ist. Dieser Befehl wird normalerweise für Subroutinen eingesetzt, die Bestandteile des Programms sind.

Dagegen spricht der JSR-Befehl Subroutinen

an, die auf festen Adressen liegen. JSR kann Unterprogramme des ROM oder Standardroutinen aufrufen, die im Speicher immer den gleichen Platz einnehmen, beispielsweise Teile des Diskettenbetriebssystems.

Wenn der Prozessor ein BSR oder JSR findet, „schiebt" er den aktuellen Wert des Befehlszählers unter Verwendung des S-Registers (Stack Pointer) auf den System-Stack. Sobald die Subroutine jedoch das S-Register für eigene Zwecke einsetzt, muß dieses Register bei der Rückkehr wieder in den ursprünglichen Zustand versetzt werden. BSR berechnet die Adresse der Subroutine und lädt sie in den Befehlszähler. Der nächste Befehl ist nun die erste Instruktion der Subroutine. Wichtig ist hierbei, daß die Subroutine mit einem Befehl und nicht mit einem Datenbyte beginnt.

Subroutinen müssen mit dem Befehl RTS (Rückkehr aus einem Unterprogramm) abschließen. Dabei wird der alte Wert des Befehlszählers wieder vom Stack „heruntergezogen" und das Programm veranlaßt, an der Stelle weiterzumachen, an der die Verzweigung eingeleitet wurde.

Unser Beispielprogramm ist komplizierter als die bislang gezeigten Module, kann aber durch den Einsatz einer Subroutine „handlicher" gestaltet werden. Das Programm durchsucht eine Tabelle mit Strings unterschiedlicher Länge und entnimmt ihr einen Wert, der einem bestimmten String zugeordnet ist. Das Format der Strings folgt dem bekannten Standard: Am Anfang steht ein Längenbyte, gefolgt von den Zeichen des Strings. Den Abschluß bildet eine 16-Bit-Adresse.

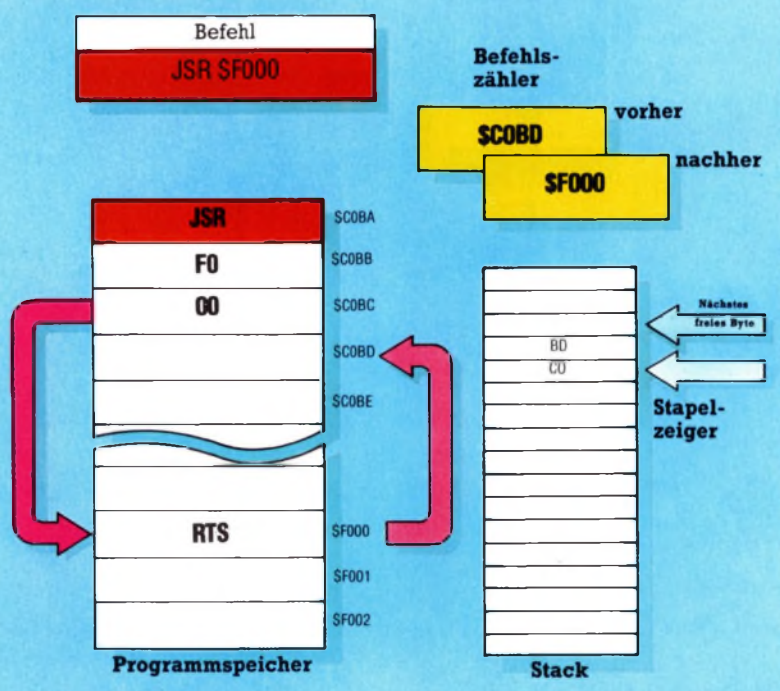
Das Ende der Tabelle wird durch einen String der Länge Null angezeigt. Das heißt, das entsprechende Längenbyte steht auf Null. In $10 ist die Anfangsadresse der Tabelle abgelegt und in $12 die Adresse des Strings, nach dem die Tabelle durchsucht werden soll. Findet sich eine Entsprechung, dann wird die Adresse in $14 gespeichert, wenn nicht, werden $12 und $14 auf Null gesetzt.

Übereinstimmungen

String-Vergleiche werden in der Programmierung oft gebraucht, speziell in der Verwaltung von String-Variablen durch BASIC-Interpreter: Alle Variablennamen werden durch Adressen ersetzt, an denen die Variablenwerte abgelegt wurden.

Unsere Programmieraufgabe läßt sich leicht in zwei Aufgabenbereiche unterteilen: Die Tabelle muß Schritt für Schritt durchsucht werden, bis entweder der String gefunden oder das Ende der Tabelle erreicht ist. Außerdem werden bei jedem Schritt zwei Strings miteinander verglichen (der gesuchte und der in der Tabelle gespeicherte).

Stacksprünge



Beim Aufruf und Rücksprung einer Subroutine (Jump and Return) wird der augenblickliche Wert des Befehlszählers gespeichert, durch die Adresse der Subroutine ersetzt und schließlich wieder in seinen ursprünglichen Zustand zurückversetzt. Der Stack ist ein Speicherbereich, in dem der Prozessor die Rücksprungadresse ablegt, der Stack Pointer (Stapelzeiger) dagegen ein 16-Bit-Register, das immer die Adresse des nächsten freien Stapelbytes enthält.

Wenn der Prozessor den Befehl JST beispielsweise an der Adresse $C0BA findet, lädt er automatisch $C0BD – die Adresse des auf JSR folgenden Befehls (ABX) – in den Befehlszähler. Danach wird die Subroutine ausgeführt, bis sie auf den RTS-Befehl (Rückkehr von der Subroutine) trifft. Nun wird die Rücksprungadresse $C0BD wieder vom Stack „gezogen" und in den Befehlszähler eingesetzt.

Dieser Vergleichsvorgang eignet sich ausgezeichnet für eine Subroutine – nicht nur weil er von dem Programm vielfach aufgerufen werden kann, sondern auch weil sich das Programm damit in recht handliche Aufgabenbereiche unterteilen läßt.

Die Subroutine benötigt vom Hauptprogramm zwei Informationen: die beiden Adressen der Strings, die miteinander verglichen werden sollen. Da die Subroutine die Strings byteweise bearbeitet, werden diese Werte am besten in die Indexregister X und Y gelegt. Die Unterroutine muß ebenfalls zwei Werte zurückgeben – einen Wert, der anzeigt, ob eine Entsprechung gefunden wurde oder nicht, und bei Übereinstimmung die Adresse des Strings.

Mit einem der Flags des Condition Code Registers kann auch ein Boolescher Parameter (wahr oder falsch) übergeben werden. Da dafür aber die Auswirkungen jedes einzelnen Flags bekannt sein müssen, zeigen wir dem Hauptprogramm mit $00 (nur Nullen) an, daß eine Entsprechung gefunden wurde, und mit $FF (nur Einsen) den umgekehrten Fall.

Um die Subroutine universell einsetzbar zu machen, wird beim Finden einer Entsprechung nicht die eigentliche String-Adresse zurückgegeben, sondern das X-Register auf eine Adresse gesetzt, die auf die gefundene Adresse zeigt. Da das X-Register diese Information beim schrittweisen Durchgehen der Tabelle automatisch erhält, sparen wir hierbei den Code für die Adressenspeicherung.

```
TABLE   EQU   $10
STRNG   EQU   $12
ADDRS   EQU   $14
        ORG   $1000
        LDU   TABLE
LOOP1   LDA   ,U
        BEQ   NTFND1
        ADDA  #3
        TFR   U,X
        LEAU  A,U
        LDY   STRNG
        BSR   COMPAR
        TSTA
        BEQ   FOUND1
        BRA   LOOP1
FOUND1  LDD   ,X
        BRA   FINSH1
NTFND1  LDD   #0
FINSH1  STD   ADDRS
        SWI
COMPAR  LDB   ,X+
        CMPB  ,Y+
        BNE   NOTEQ
LOOP2   LDA   ,X+
        CMPA  ,Y+
        BNE   NOTEQ
        DECB
        BGT   LOOP2
        CLRA
        BRA   FINSH2
NOTEQ   LDA   #$FF
FINSH2  RTS
        END
```

Anfang des Hauptprogramms

Tabellenanfang in U ablegen

Längenbyte holen

GOTO Tabellenende, falls Null

Eins für das Längenbyte und Zwei für die Adresse zu der Stringlänge addieren

Anfang des Tabellenstrings in X laden

U auf den nächsten Eintrag setzen

Y zeigt auf den Anfang des zu suchenden Strings

Die beiden Strings vergleichen

Prüfen, ob sie gleich sind

Wenn sie gleich sind, GOTO FOUND1

ELSE nächsten Tabelleneintrag holen

Falls FOUND1, soll X auf die gewünschte Adresse zeigen

Falls nicht FOUND1, Adresse auf Null setzen

Adresse speichern

Ende Hauptprogramm

Anfang der Subroutine

Die Längenbytes holen und X und Y auf die ersten Zeichen stellen

Wenn Längen nicht identisch: GOTO NOTEQ

Nächstes Zeichen des Tabellenstrings holen

Mit dem nächsten Zeichen des gesuchten Strings vergleichen

Stop, falls nicht identisch

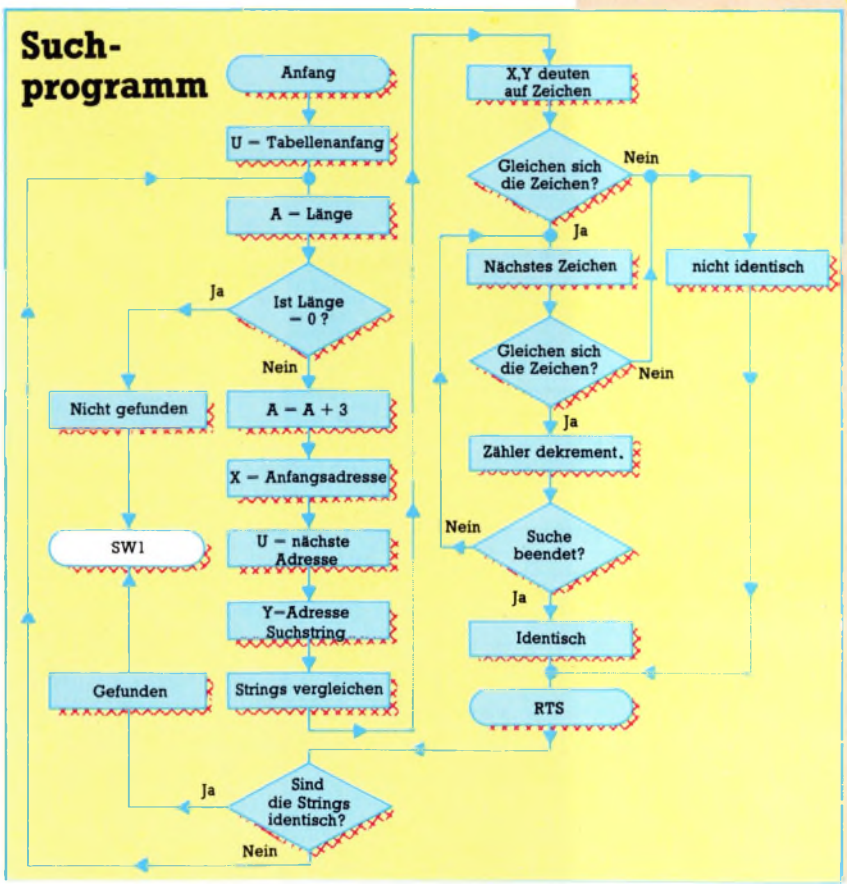
Sonst Eins vom Positionszeiger abziehen

Nächstes Zeichen holen

A auf Null setzen, um anzuzeigen, daß die Strings identisch sind

oder auf Eins, wenn sie nicht identisch sind

Zurück zum aufrufenden Programm



Große Sprünge

Nachdem in der letzten Folge untersucht wurde, wie die indizierte Adressierung des 6809-Prozessors funktioniert, sehen wir uns heute die indirekte Adreßmethode an und entwickeln ein Programm, das Zeichen auf dem Bildschirm darstellt.

Die indirekte Adressierung läßt sich mit fast allen anderen Adreßarten einsetzen. Dabei wird eine Adresse berechnet und der Inhalt dieser Adresse sowie die darauffolgende Speicherstelle als aktuelle Adresse angesehen. Aus dieser endgültigen Adresse werden dann die Daten geladen.

Bei den folgenden Werten:

Adresse	Inhalt
3000	40
3001	0A
400A	F2

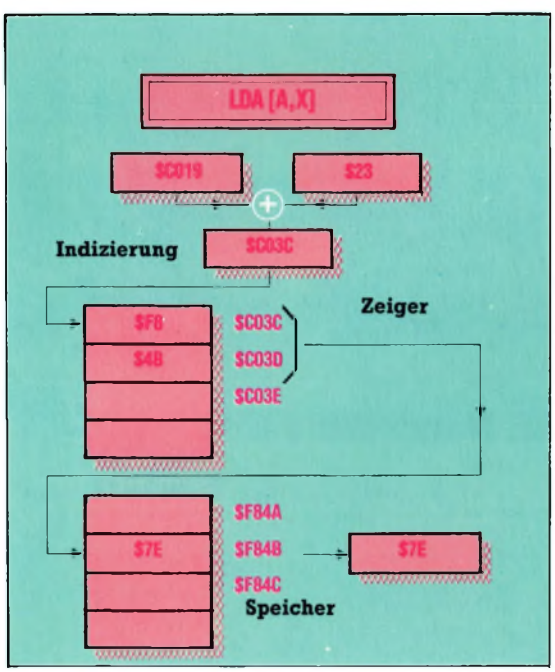
lädt der direkte Ladebefehl LDA $3000 den Wert $40 in den Akkumulator A – die aktuelle Adresse ist dabei $3000. Bei der indirekten Adressierung steht der Operand in eckigen Klammern. LDA [$3000] lädt daher den Wert $F2 in A, wobei die aktuelle Adresse von dem Wert dargestellt wird, der in $3000 und $3001 gespeichert ist – in diesem Fall $400A. Der Inhalt von $3000 und $3001 bildet einen Zeiger (oder Vektor) auf die aktuelle Adresse $400A. Beachten Sie, daß die Adressen des 6809 das Format „höherwertiges Byte vor niederwertigem Byte" (auch hi-lo genannt) haben: $40 wird daher in $3000 gespeichert und $0A in $3001. Der Z80 von Zilog und der 6502 von MOS Tech verwenden das umgekehrte Format – $0A (das niederwertige Adreßbyte) wird in $3000 gespeichert, $40 (das höherwertige Byte) in $3001.

Die indirekte Adressierung läßt sich gut mit der indizierten Adressierung kombinieren. Der Befehl LDA [A,X] (indirekt indiziert) berechnet eine Adresse, indem er den Inhalt von A und X addiert und dann den 16-Bit-Wert dieser und der darauffolgenden Speicherstelle als aktuelle Adresse einsetzt, dessen Inhalt in A geladen wird.

Tatsächlich benutzt man beim 6809 die indirekte Adressierung jedoch weitaus weniger als bei den meisten anderen Prozessoren (Programme des Z80 und 6502 verwenden sie oft), da er über viele Möglichkeiten der indizierten Adressierung verfügt. Es gibt jedoch Aufgaben, die durch die indirekte Adressierung sehr vereinfacht werden, unter anderem der Einsatz von Peripheriegeräten (auf den wir in einer der nächsten Folgen ausführlich eingehen werden). Die Motorola-Prozessoren setzen im Gegensatz zu den 8080- und 8060-Chip-Familien von Intel für die Ein- und Ausgabe eine Memory-Map ein. Die Kommunikationsregister werden dabei in der Memory-Map des Systems abgelegt und lassen sich wie normale Speicherstellen ansprechen. Ein Steuermodul – beispielsweise eine Druckroutine – braucht die Adresse des entsprechenden Schnittstellenregisters. Wenn dieses Register nun an eine andere Stelle der Memory-Map verlegt wurde, oder nicht nur ein Gerät dieser Art existiert, ist es einfacher, die Speicherstelle mit der Adresse des Kommunikationsregisters (den Zeiger) zu ändern als jede einzelne Registeradresse. Die Routine bezieht sich daher über den Zeiger indirekt auf das Peripheriegerät.

Wenn sich die Bezugsadressen eines Programms verändern können, sind Adreßzeiger sehr praktisch, da bei einer Veränderung der aktuellen Adresse lediglich der Inhalt des Zeigers geändert werden muß.

Bei dieser Technik wird viel mit einer Struktur gearbeitet, die „Sprung-Tabelle" heißt und einfach eine Liste von Zeigern umfaßt. Jedes Betriebssystem enthält eine Reihe von Routinen, die grundlegende Funktionen ausführen – beispielsweise Zeichen von der Tastatur lesen oder auf dem Bildschirm anzeigen. Viele Maschinencodeprogramme rufen diese Routinen über die Sprung-Tabelle auf. Dabei können die Routinen, auf die die Werte der Sprung-Tabelle zeigen, geändert oder verlegt werden.



Das Argument [A,X] des LDA-Befehls ist in eckige Klammern eingeschlossen. Das bedeutet, daß der Inhalt von X (hier $C019) mit dem Inhalt von A ($23) addiert wird und so die 16-Bit-Adresse ($C03C) ergibt. Dieses und das nächste Byte ($C03C) sind nun der Zeiger auf die zu ladende Adresse ($F84B), deren Inhalt schließlich in A übertragen wird. Da X vor dem indirekten Zugriff zu A addiert wird, nennt man diese Adreßart „indirekt vorindiziert". Bei dem Gegenstück – der „Nachindizierung" – wird zunächst die indirekte Adresse berechnet und danach die Indizierung durchgeführt.

Viele Betriebssysteme besitzen nur eine Einsprungadresse für Unterroutinenaufrufe. Dabei wird eins der CPU-Register mit dem Funktionscode geladen, der das aufzurufende Modul angibt. Dieser Code ist der Index (oder Offset) für den entsprechenden Vektor der Sprung-Tabelle, dessen Adresse die gewünschte Routine anspricht.

Nehmen wir als Beispiel ein ROM mit vier KByte, die bei $F000 liegen. Die ersten 256 Byte ($F000 bis $F0FF) enthalten eine Tabelle mit bis zu 128 Adressen für Subroutinen, die irgendwo im ROM untergebracht sind. Die Einsprungroutine liegt bei $F100 und erwartet im Akkumulator B einen Funktionscode im Bereich von 0 bis 127. Mit diesem Code übergibt die Einsprungroutine die Steuerung an die entsprechende Unterroutine und nach deren Ausführung wieder zurück an das aufrufende Programm. Der Aufruf von Funktion 1 sieht folgendermaßen aus:

```
LDB #1      den Funktionscode in B laden
JSR $F100   Einsprungroutine aufrufen
```

Die Einsprungroutine selbst lautet:

```
LDX $F000   Anfangsadresse der Sprung-Tabelle
LSLB        B um eine Stelle nach links verschieben
            (entspricht einer Multiplikation von B mit
            zwei), da jeder Tabelleneintrag zwei
            Byte lang ist. Der Zeiger für den Funk-
            tionscode 1 befindet sich in $F002 und
            $F003, während der Zeiger für Code 2
            auf $F004 und $F005 liegt etc.
BRA [B,X]   Steuerung an die Adresse übergeben,
            die an der von B angegebenen Tabellen-
            position gespeichert ist.
```

Beachten Sie, daß die Übergabe an die Routine mit BRA (oder JMP) und nicht mit BSR (oder JSR) geschieht. Damit gibt das RTS am Ende der Betriebssystemroutine die Steuerung direkt an das aufrufende Programm zurück, ohne erst die Einsprungroutine ansprechen zu müssen.

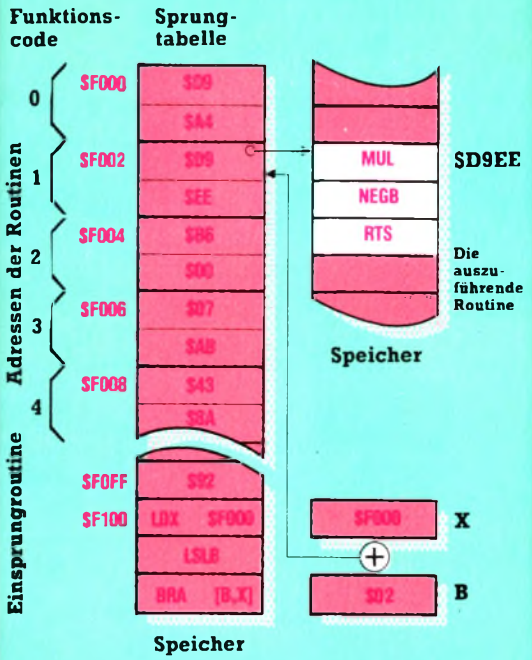
Unser nächstes Beispiel zeigt die indirekte Adressierung einer Bildschirmanzeige, die mit einer Memory-Map arbeitet. Auf vielen Micros belegt der Bildschirmspeicher einen Teil des Arbeitsspeichers, der für höhere Anzeigegeschwindigkeiten auch direkt adressiert werden kann. Nehmen wir der Einfachheit halber an, daß der Bildschirm einen Speicherblock von $E000 bis $E3FF belegt und 16 Zeilen mit je 64 Zeichen enthält. Die Position des Cursors (ein 16-Bit-Wert dieses Adreßbereichs) befindet sich in $E400. Die erste Subroutine löscht die Bildschirmpositionen zunächst durch Einsetzen von Leerzeichen (ASCII-Code 32). Die zweite Subroutine stellt nun das in A gespeicherte Zeichen auf der aktuellen Cursorposition dar, wenn kein Return (ASCII-Code 13) eingegeben wurde. Beim Drücken der Return-Taste wird der Rest der Zeile gelöscht und der Cursor auf den Anfang der nächsten Zeile gesetzt. In diesem Beispiel stellt die Unterstreichung ("_") den Cursor dar.

```
SPACE   EQU 32      ASCII-Code für Leerzeichen
CR      EQU 13      ASCII-Code für Return
HOME    EQU $E000   Anfang des Bildschirmspeichers
LENGTH  EQU 1024    Größe des Bildschirmspeichers (16
                    Zeilen × 64 Zeichen = 1024)
```

Obwohl die indirekte Adressierung für Computerabläufe eine große Bedeutung hat, gibt es für diese Technik im täglichen Leben nur wenige Beispiele. Eine brauchbare Analogie ist das Eurosignal, bei dem eine Person über ein kleines tragbares Empfangsgerät angerufen werden kann. Will jemand diese Person sprechen, ruft er sie nicht direkt an (da er ihren genauen Aufenthaltsort nicht kennt), sondern die Zentralstation, die die gewünschte Person dann per Funk erreicht. Dieser einfache und flexible Service der Zentrale ist eine Form der „indirekten Adressierung".

```
CURSOR    EQU $E400      $E400 und $E401 zeigen gemeinsam auf die aktuelle Adresse des Cursors im Bereich des Bildschirmspeichers
          ORG $1000
CURCHR    FCB 95         Unterstreichung (ASCII 95)
*** Subroutine zum Löschen des Schirms ***
          LDA #SPACE     Leerzeichen in A
          LDX #HOME      Cursor auf den Bildschirmanfang setzen
          STX CURSOR     Die aktuelle Cursorposition in CURSOR speichern (d. h. $E400 und $E401)
          LDB #LENGTH    Größe des Bildschirms in B
LOOP1     STA[CURSOR]    Leerzeichen auf der aktuellen Cursorposition speichern
          INC CURSOR     Cursorposition inkrementieren
          DECB           Bildschirmspeicher, der zwischen der Cursorposition und dem Ende des Bildschirmspeichers liegt, dekrementieren
          BGT LOOP1      Nächstes Leerzeichen, bis das Ende des Bildschirmspeichers erreicht ist
          STX CURSOR     Cursor auf die Home-Position stellen
          LDA CURSOR     ASCII-Code des Cursorzeichens in A
          STA[CURSOR]    Cursorzeichen in der aktuellen Cursorposition speichern
          RTS
** Subroutine zum Anzeigen des in A gespeicherten Zeichens ***
          CMPA SPACE     In ASCII ist das Leerzeichen das erste darstellbare Zeichen
          BLT NOTP       Falls der Akkumulator einen ASCII-Wert unter 32 enthält, kann das Zeichen nicht gedruckt werden, dann GOTO NOTP
          STA[CURSOR]    In der aktuellen Cursorposition speichern
          INC CURSOR     Die Cursorposition inkrementieren
CHKEOS    LDX #HOME      Auf Bildschirmende überprüfen
          LEAY #LENGTH,X Bildschirmende in Y
          CMPY CURSOR    Falls die Cursorposition über das Bildschirmende hinausgeht, dann ...
          BGT FINISH     haben wir das Ende des Bildschirms erreicht, deshalb GOTO FINISH
*** Subroutine für das Rollen der Anzeige ***
SCROLL    LEAY 64,X      Y ist die Länge einer Zeile von X (Bildschirmende) entfernt
          LDB #LENGTH    Berechnen, um wieviel die Anzeige gerollt werden muß
          SUBB #64       64 von Length abziehen
LOOP2     LDA,Y+         Die Zeichen um eine Zeile zurückbewegen (automatische Inkrementierung)
          STA,X+
          DECB
          BGT LOOP2      Schleife durchlaufen, bis das Rollen beendet ist
          LDD CURSOR     CURSOR an den Anfang der letzten Zeile
          SUBD #64
          STD CURSOR
          BRA FINISH
*** Subroutine zum Abfangen des Return ***
NOTP      CMPA #CR       Ist das nicht darstellbare Zeichen ein Return?
          BNE FINISH     Falls nicht – ignorieren
          LDD CURSOR     Anfang der nächsten Zeile bestimmen (binäre AND-Maske)
          ANDB
          #%11100000
          ADDD #64
          STD CURSOR
          BRA CHKEOS     Auf Bildschirmende überprüfen
FINISH    LDA CURSOR     Cursorzeichen in A
          STA[CURSOR]    Aktuelle Cursorposition speichern
          RTS
```

Sprungtabelle



Die im Bild gezeigte Sprungtabelle enthält 128 Zwei-Byte-Adreßzeiger, die von $F000 bis $F0FF gespeichert sind. Jeder dieser Zeiger bezeichnet die Anfangsadresse einer Routine, die sich im Speicher befindet. Für die Ausführung dieser Routinen muß nur der Akkumulator B mit dem Funktionscode (etwa $01) geladen werden, der die gewünschte Routine bezeichnet (hier $D9EE), und mit JSR zur „Einsprungroutine" bei $F0FF verzweigt werden. Wenn es sich bei diesen Routinen um Teile eines im ROM abgelegten Betriebssystems handelt, sollte das Handbuch Auskunft geben, welche Routinen sich hinter den Funktionscodes verbergen.

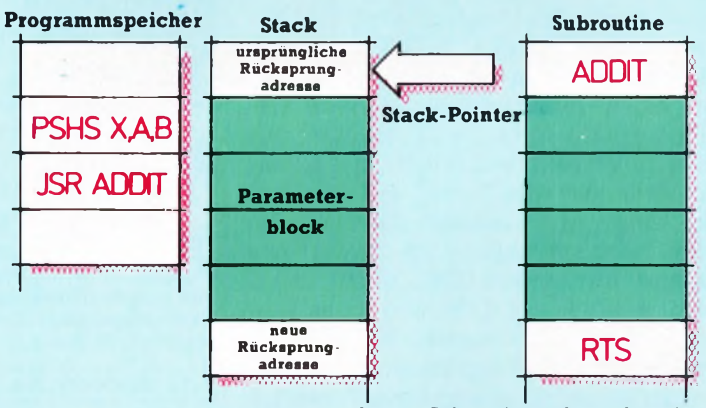
Die Einsprungroutine multipliziert den Funktionscode mit Zwei, nimmt diese Zahl als Offset für die Anfangsadresse der Tabelle und findet so den Adreßzeiger der gewünschten Routine. Der Zeiger für Routine $01 befindet sich bei $F002 (=$F000 + 2 * $01), der Zeiger auf Routine $02 bei $F004 (=$F000 + 2 * $02) etc. Der Zeiger wird dann von der Einsprungroutine in einen indirekten Verzweigungsbefehl eingesetzt, der bei $D9EE die Steuerung an die Routine übergibt. Beachten Sie, daß die Einsprungroutine die angesprochene nicht aufruft, sondern darauf verzweigt. Damit übergibt RTS die Steuerung an das aufrufende Programm zurück.

Mit Sprungtabellen können Programme ohne Änderungen ablaufen, selbst wenn einzelne Routinen geändert oder umgestellt wurden. Selbst bei Änderungen des Betriebssystems bleiben die Funktionscodes und die Adresse der Einsprungroutine die gleichen, während der Inhalt der Sprungadressen verändert werden kann.

Tiefstapeln

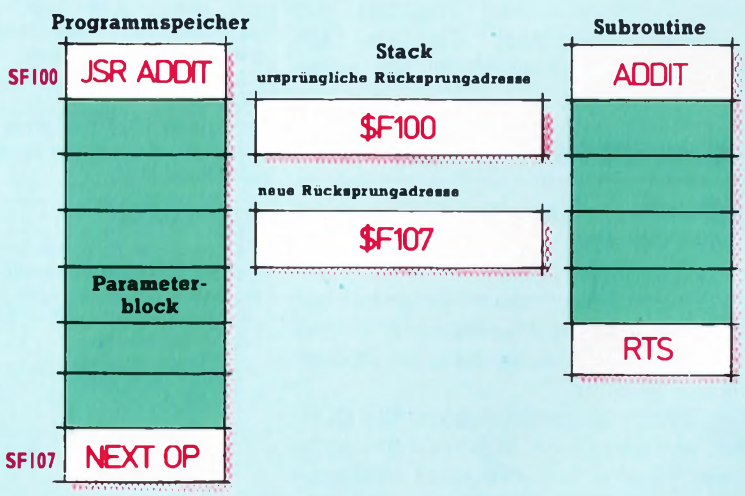
Zwar haben wir uns schon ausführlich mit den Adressierungsarten des 6809 beschäftigt, doch gibt es einige Varianten, die noch untersucht werden müssen, wie beispielsweise der Einsatz des Befehlszählers als Index.

Parameter übergeben



Parameter werden an Subroutinen übergeben, indem sie in Register geladen und dann auf den Stack geschoben werden. Die Subroutine zieht sie wieder vom Stack herunter. Dabei ist zu beachten, daß die Rücksprungadresse nach dem Laden an die richtige Stelle des Stacks gesetzt wird. Andernfalls verursacht der Stack einen Overflow.

Parameter einsetzen



Praktischer ist es, die zu übergebenden Parameter direkt hinter dem Aufruf (JSR) der Subroutine in das Programm einzusetzen. Die Subroutine kann nun die auf dem Stack gespeicherte Rücksprungadresse als Basisadresse des Parameterblocks einsetzen und mit der indizierten Adressierung darauf zugreifen. Vor dem RTS-Befehl muß die Rücksprungadresse verändert werden, damit sie auf den nächsten auszuführenden Befehl zeigt und nicht auf den Anfang des Parameterblocks.

Bisher wurden die beiden Register S und U (die Stack-Pointer) nur als zusätzliche Indexregister eingesetzt, und der sogenannte „Hardwarestack" zur Speicherung der Rücksprungadressen aus Subroutinen wurde nur nebenbei erwähnt. In diesem Artikel gehen wir daher zunächst genauer auf die Arbeitsweise des Stacks ein.

Der Stack läßt sich als eine spezielle „Liste" beschreiben. Wie Sie bereits wissen, besteht eine Liste aus einer Folge von Datenelementen. Diese Folge kann nach bestimmten Eigenschaften der Daten geordnet werden (eine Zahlenserie in numerischer Reihenfolge, eine Zeichenkette in alphabetischer Sortierung etc.). Eine Anordnung entsteht aber auch durch die Reihenfolge, in der Datenelemente an die Liste angefügt werden. Bei all diesen Listen hat der Inhalt des folgenden und des vorhergehenden Datenelements große Bedeutung, insbesondere das erste (der „Listenkopf") und das letzte Element (der „Abschluß").

Wichtig ist ebenfalls die „dynamische Datenstruktur" einer Liste. In allgemeingültig definierten Listen können Datenelemente beliebig eingefügt oder herausgenommen werden. Eine Liste wird zu einem „Stack" (Stapel), wenn Daten nur an einem Ende herausgenommen oder angefügt werden können. Jedes neu hinzukommende Datenelement wird zum Listenkopf und kann auch nur als erstes Element wieder herausgenommen werden.

Gestapelte Information

Die Bezeichnung Stack gibt schon Auskunft über seine Funktionsweise. Stellen Sie sich einen Tellerstapel in einer Kantine vor: Die Teller werden immer nur vom oberen Ende des Stapels genommen, und neue Teller werden auf die Spitze des Stapels gesetzt. Es wäre zwar möglich, auch aus der Mitte des Stapels Teller herauszunehmen oder dort einzusetzen, doch würde der Vorgang dadurch unnötig kompliziert. Es ist jedoch möglich, jedes Element des Stapels zu überprüfen. Das Hinzufügen und Wegnehmen wird „auf den Stack schieben" (PUSH) bzw. „herunterziehen" (PULL oder POP) genannt.

Bei der Arbeit eines Stacks können zwei extreme Situationen eintreten: Erstens, der Stack

ist leer (wenn der nächste Stack-Befehl Daten anfügt, entsteht kein Problem, wenn nicht, gibt es Schwierigkeiten), und zweitens, der Stack fließt über. In unserer Analogie bedeutet dies: Wenn der Tellerstapel die Decke erreicht, läßt sich kein Teller mehr daraufsetzen. Diese beiden Extremsituationen heißen „Underflow" (Unterfluß) und „Overflow" (Überfluß).

Stacks können auf vielerlei Weise eingesetzt werden (in BASIC beispielsweise als Arrays). Für unsere Zwecke benötigen wir einen frei verfügbaren Speicherblock und ein Register als „Stack-Pointer". Dieser Pointer enthält die Adresse des Listenkopfes. Der Speicherstack kann jedoch nicht von außen untersucht werden, da sich eine Speicheradresse mit Stack-Element nicht von Speicherstellen mit anderen Daten unterscheidet. Wichtig ist auch, daß Daten nicht wirklich aus dem Speicher in ein Register „geladen", sondern nur kopiert werden. Sie werden nicht wirklich vom Stack „gezogen", sondern es wird nur der Pointer auf eine andere Adresse gesetzt.

Wie ein Verschiebebahnhof

Im Gegensatz zu unserem Tellerstapel „wächst" ein Speicher-Stack des 6809-Systems abwärts. Je mehr Elemente auf den Stack geschoben werden, desto niedriger wird die Adresse des Stack-Pointers – er „wächst gegen Null".

Zwei Befehle des 6809 lösen Stack-Vorgänge aus: PSH schiebt Daten auf den Stack, und PUL zieht sie wieder herunter. Da für diese Vorgänge die beiden Pointer S und U eingesetzt werden können, stehen die Befehle PSHS, PULS, PSHU und PULU zur Verfügung.

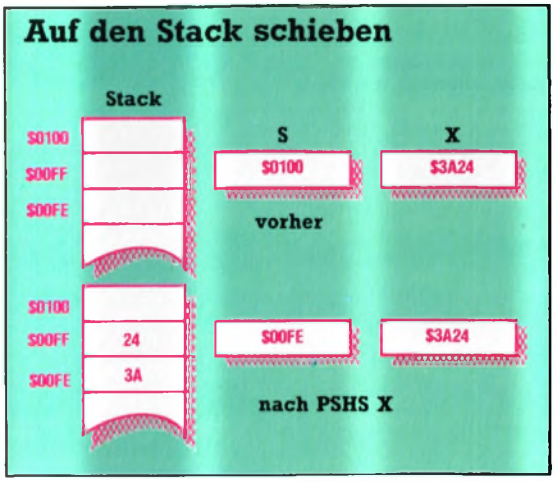
Um die Adresse des nächsten freien Stack-Speichers zu erhalten, dekrementiert der Befehl PSHS X zuerst den Stack-Pointer S um zwei (oder eins, wenn ein Acht-Bit-Register gespeichert wird) und legt dann den Inhalt von X auf dieser Adresse ab. Die erste Abbildung veranschaulicht diesen Vorgang. Beachten Sie, daß der 6809 das Adreßformat hi-lo einsetzt. Das höherwertige Byte ($3A) von X ist in $00FE gespeichert, also in einer niedrigeren Speicheradresse als das in $00FF untergebrachte niederwertige Byte ($24). Wenn Sie mit einem Assembler arbeiten, ist die In- oder Dekrementierung des Stack-Pointers nicht wichtig, da das Programm alle notwendigen Speichervorgänge berücksichtigt.

Der Befehl PULS X bewirkt das genaue Gegenteil: Der 16-Bit-Wert der in S gespeicherten Adresse wird in X geladen und der Inhalt von S dann um zwei inkrementiert. Das zweite Diagramm zeigt diese Veränderungen.

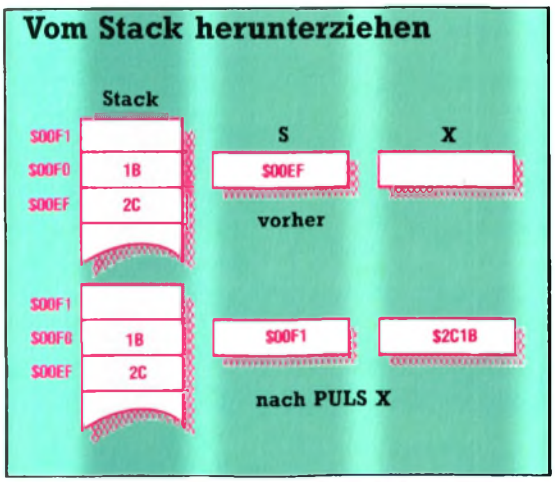
Es lassen sich auch mehrere Register gleichzeitig heraufschieben oder herunterziehen, beispielsweise:

PSHS X,Y,U,A

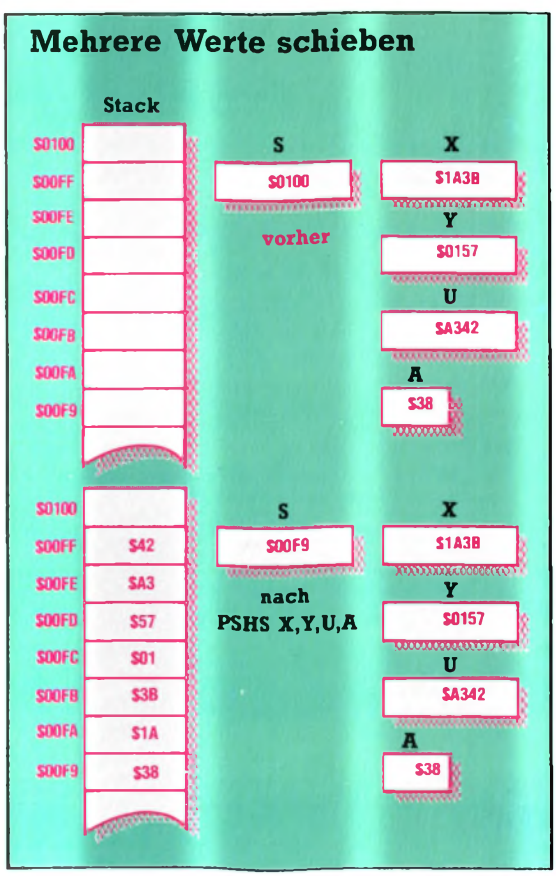
Die Anordnung der Register hat in diesem Fall



Der Stack-Pointer des 6809 adressiert immer die „Spitze" des Stacks – das heißt das Byte, in das zuletzt geschrieben wurde. Bei der Ausführung von PSHS X wird daher S um zwei dekrementiert und zeigt so auf die neue Stackspitze. Der Inhalt von X wird dann im Format hi-lo auf dieser Adresse gespeichert. Beachten Sie, daß der Stack „gegen Null wächst", das heißt, der Stack-Pointer zeigt beim Wachsen des Stacks auf immer niedrigere Adressen.



Bei der Ausführung von PULS X wird der Inhalt der zwei Bytes beim aktuellen Stack-Pointer nach X kopiert. Danach wird S um zwei dekrementiert, damit der Pointer auf die neue Stack-Spitze zeigt.



Bei einem Stack-Befehl, der mehrere Werte gleichzeitig umfaßt, werden die angesprochenen Register immer in der gleichen vorbestimmten Folge abgelegt: PC, U oder S, Y, X, DP, B, A, CC. Wenn daher PSHS X,Y,U,A ausgeführt wird, wird zuerst U gespeichert, dann Y, X und A.

Addieren mit mehrfacher Präzision

Die beiden Programmteile zeigen zwei Möglichkeiten, wie sich mit Stacks Additionen mit mehrfacher Präzision ausführen lassen. Das erste Beispiel speichert seine Parameter gleich hinter dem Subroutinenaufruf. Dabei werden die beiden Zahlen (in $100 und $104) addiert und das Ergebnis in $108 gespeichert:

```
        BSR   MPADD
        FCB   4          Länge jeder Zahl (in Bytes)
        FDB   $100       Adresse der ersten Zahl
        FDB   $104       Adresse der zweiten Zahl
        FDB   $108       Adresse des Ergebnisses
        ORG   $1000
MPADD   PSHS  X,Y,U,A,B,CC   Alle Register sichern. Dieser Befehl schiebt neun Bytes auf den Stack
        LDU   9,S        Dieser Befehl lädt die mit BSR auf den Stack geschobene PC-Rücksprungadresse in U
        PULU  X,Y,B      Die hinter dem Subroutinenaufruf gespeicherten Daten werden wie ein Stack behandelt, die Adressen der zwei Zahlen in X und Y und die Länge in B gespeichert
        LDU   ,U         Die Adresse des Ergebnisses wird in U abgelegt
        ANDCC #%11111110 Übertragsflag löschen
LOOP    LDA   ,X+        Ein Byte auf das andere addieren (mit Übertrag)
        ADCA  ,Y+
        STA   ,U+        Ergebnis speichern
        DECB             Prüfen, ob Addition beendet
        BNE   LOOP       Falls nicht, LOOP
        LDU   9,S        Sonst Rücksprungadresse holen
        LEAU  7,U        und sieben addieren, damit die
        STU   9,S        sieben Parameterbytes übersprungen werden
        PULS  PC,U,X,Y,A,B,CC   Alle Register wieder in ihren ursprünglichen Zustand versetzen und statt eines RTS-Befehls den PC setzen
```

Das zweite Beispiel führt den gleichen Vorgang aus, schiebt die Parameter jedoch auf den Stack. Die aufrufende Routine sieht folgendermaßen aus:

```
        LDU   #$108      Adresse des Ergebnisses
        LDX   #$100      Adresse des Operanden
        LDY   #$104
        LDA   #4         Länge
        PSHS  U,X,Y,A    Parameter auf den Stack schieben
        BSR   MPADD
        ORG   $1000
MPADD   PSHS  X,Y,U,A,B,CC   Alle Register sichern
        LEAU  11,U       Das Sichern der Register belegt neun Bytes und die Rücksprungadresse zwei Bytes – insgesamt elf. U zeigt nun auf die Parameter
        PULU  X,Y,B
        ANDCC #%11111110
LOOP    LDA   ,X+        Gleicher Ablauf wie im ersten Beispiel
        ADCA  ,Y+
        STA   ,U+
        DECB
        BNE   LOOP
        PULS  PC,U,X,Y,A,B,CC
```

keine Bedeutung, da immer in der gleichen Reihenfolge gespeichert wird: PC (der Befehlszähler), U oder S, Y, X, DP (das Seitenregister für die direkte Adressierung), B, A und CC (das Condition Code Register). Das Herunterziehen geschieht natürlich in umgekehrter Reihenfolge. Lediglich der eigene Stack-Pointer S oder U kann nicht auf den zugehörigen Stack geschoben werden.

Stacks werden in der Programmierung als schnelle Zwischenspeicher eingesetzt. Ihre größte Bedeutung haben sie jedoch bei der Behandlung von Interrupts (auf die wir später genauer eingehen) und Subroutinen. Wir haben bereits gezeigt, wie der Aufruf einer Subroutine den Inhalt des Befehlszählers automatisch auf den Stack schiebt und beim Rücksprung wieder herunterzieht (RTS entspricht PULS PC). Mit beiden Stacks – besonders aber mit S – lassen sich Parameter an Subroutinen übergeben.

Ohne Komplikationen

Diese Aufgabe haben wir bisher zwar mit Registern erledigt, doch hat diese Technik zwei schwache Punkte. Erstens kann der Fall eintreten, daß mehr Parameter übergeben werden müssen, als Register zur Verfügung stehen, und zweitens kann es Komplikationen geben, wenn die aufgerufene Routine ein Register mit einem Parameter verwendet, der noch gebraucht wird. Für die Parameterübergabe werden häufig die beiden folgenden Techniken eingesetzt:

1) Die Daten werden unmittelbar nach Aufruf der Subroutine mit FCB-, FDB- oder FCC-Anweisungen in der Mitte des Programms gespeichert. Der Wert des Befehlszählers, den der JSR-Befehl auf den Stack geschoben hat, enthält die Adresse der ersten Speicherstelle (der Befehlszähler zeigt immer auf das Byte, das der aktuellen Instruktion folgt). Die Daten lassen sich nun mit geeigneten Offsets lesen. Unser erstes Programmbeispiel demonstriert diese Technik. Dabei muß jedoch beachtet werden, daß RTS die Steuerung an den nächsten gültigen Befehl übergibt und nicht an ein Datenelement.

2) Die Daten werden in Register geladen und vor Aufruf der Subroutine auf den Stack geschoben. Die Subroutine zieht die Daten wieder herunter und verarbeitet sie. Dabei ist zu beachten, daß der Stack-Pointer beim Rücksprung (RTS) wieder auf den zuvor gespeicherten Wert des Befehlszählers zeigt.

Bei beiden Methoden werden S und U als Indexregister und auch als Stack-Pointer eingesetzt. Dadurch kann mit der indizierten Adressierung leicht auf die Stacks zugegriffen werden, und die Stack-Daten lassen sich auch leicht wieder vom Stack herunterziehen. Ferner ist sichergestellt, daß der Stack beim Rücksprung die korrekten Daten enthält.

Ein- und Ausgabe

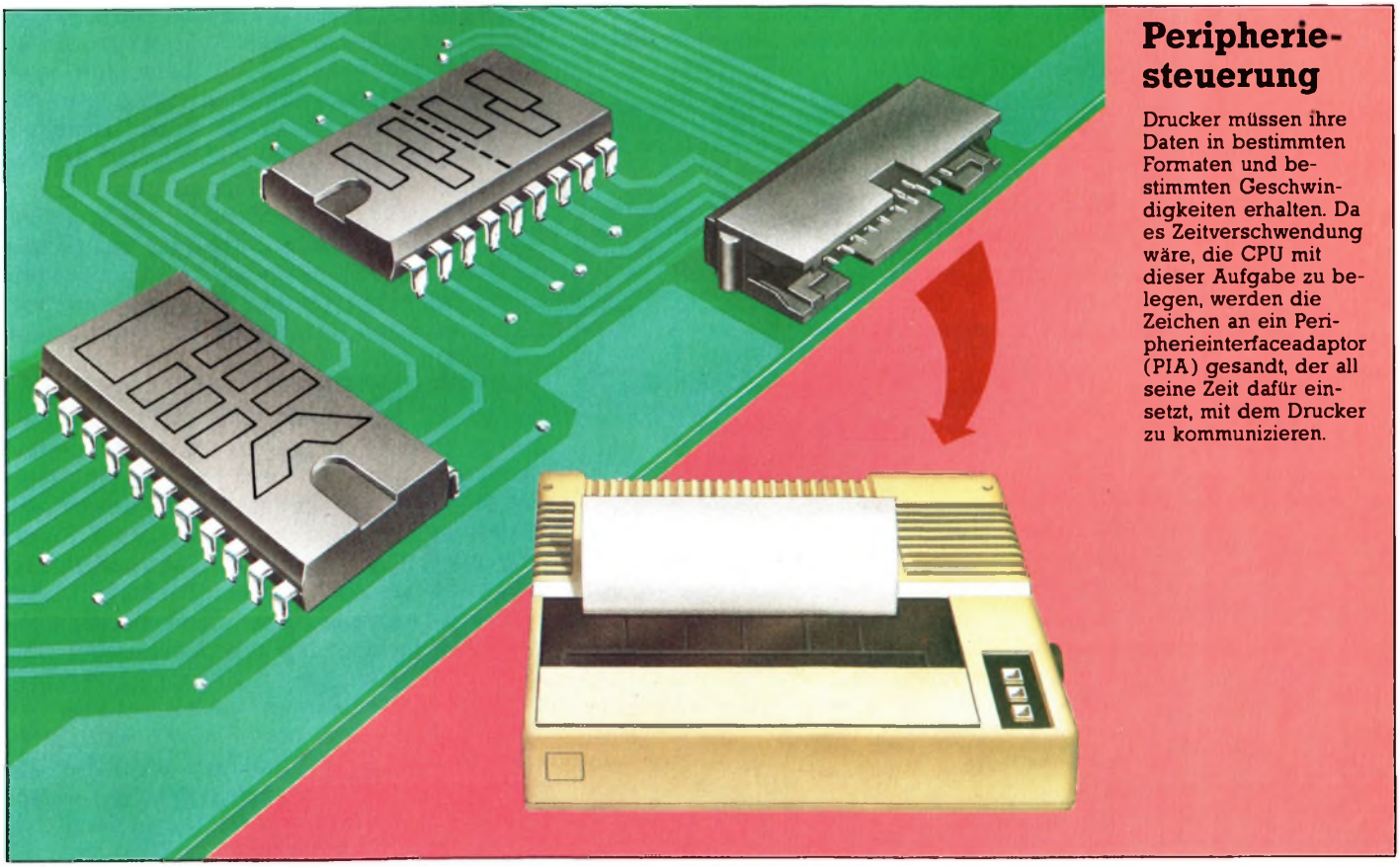
Die Steuerung von Ein- und Ausgabe ist ein wichtiger Aspekt der Assemblerprogrammierung. Wir sehen uns an, was in den beiden Schnittstellenchips 6820-PIA und 6850-ACIA vorgeht.

Der 6809-Prozessor hat wie der 6502 keinen getrennten Adreßraum für die Ein- und Ausgabe und kennt dafür auch keine speziellen Befehle. Die Ein- und Ausgabechips liegen im normalen Adreßbereich und werden wie Speicherstellen angesprochen. Diese Methode ist zwar schnell und einfach, blockiert aber einen Adreßbereich. Der 6809 kann daher trotz seines für 64 KByte ausgelegten Adreßbusses maximal nur 56 KByte einsetzen – ohne die Speicherverwaltung für Hard- und Software.

Zwar lassen sich einige Ein- und Ausgabegeräte direkt an den Systemdatenbus anschließen, doch sitzt normalerweise ein Schnittstellenchip dazwischen. Diese Chips sind hochentwickelte Schaltungen. Üblicherweise werden Chips der gleichen Prozessorfamilie eingesetzt, da der Anschluß und die Steuerung darauf abgestimmt sind. Mit dem 6809 wird für die parallele Ein- und Ausgabe meistens der 6820- (oder 6821-) PIA genommen und für die serielle E/A der 6850-ACIA. Beide Chips besitzen eine Reihe von Registern, die wie normale Speicherstellen angesprochen werden. Es gibt drei Registertypen:

- Steuerregister: In diese Register kann nur geschrieben werden. Die gespeicherten Werte programmieren den Chip für den gewünschten Vorgang, zum Beispiel zur Festlegung der Baud-Rate.
- Statusregister: Diese „Read-Only"-Register zeigen den Status des Chips an. Sie geben Auskunft darüber, ob Eingaben empfangen wurden, ob die letzte Ausgabe schon gesandt wurde oder ob Fehler aufgetreten sind.
- Datenregister: Hier liegen die Ein- und Ausgabedaten. Sie können als Schreib/Lese-Speicher funktionieren oder auch als reiner Lese- bzw. Schreibspeicher dienen.

Um Speicherplatz zu sparen, verwenden oft mehrere Register die gleiche Adresse. Wenn beispielsweise ein Statusregister und ein Steuerregister auf einer Adresse liegen, wird beim



Peripheriesteuerung

Drucker müssen ihre Daten in bestimmten Formaten und bestimmten Geschwindigkeiten erhalten. Da es Zeitverschwendung wäre, die CPU mit dieser Aufgabe zu belegen, werden die Zeichen an ein Peripherieinterfaceadaptor (PIA) gesandt, der all seine Zeit dafür einsetzt, mit dem Drucker zu kommunizieren.

Lesen das Statusregister angesprochen und beim Schreiben das Steuerregister. Ebenso teilen sich auch die Datenregister der Ein- und Ausgabe identische Adressen.

Der 6820-PIA enthält sechs Register und belegt vier aufeinanderfolgende Speicherbytes. Dabei versorgt der Chip zwei voneinander unabhängige Ausgänge, die beide je drei Register einsetzen. Die Peripherieseite des Chips hat für jeden Ausgang acht Datenleitungen und zwei Steuerleitungen zur Verfügung. Die beiden mit den entsprechenden Peripherieleitungen verbundenen Steuerleitungen geben Auskunft über den Status des Chips. Die Steuerleitung 1 ist für eingehende Signale reserviert, die Steuerleitung 2 kann Daten senden und empfangen.

Die drei Register haben folgende Aufgaben:

- Das Datenregister, dessen Bits sich einzeln setzen lassen, dient für die Ein- und Ausgabe.
- Das Datenrichtungsregister bestimmt, welche Bits des Datenregisters für die Eingabe und welche für die Ausgabe eingesetzt werden. Dafür werden die entsprechenden Bits des Richtungsregisters auf Eingabe (0) oder Ausgabe (1) gesetzt.
- Das dritte Register dient als Steuerungs- und Statusregister.

Das Datenrichtungsregister und das Datenregister liegen auf derselben Adresse. Ein Bit im Steuerregister bestimmt, welches der beiden Register dieser Adresse gerade angesprochen wird. Die nebenstehende Tabelle gibt die Offsets für die Registeradressen von der Basisadresse des Chips aus an.

Register	Offset
Daten A	0
Datenrichtung A	0
Steuer/Status A	1
Daten B	2
Datenrichtung B	2
Steuer/Status B	3

Die Bits des Steuer-/Statusregisters haben folgende Funktionen:

Bit	Funktion
7	Statusbit für Steuerleitung 1. Wird bei Empfang auf 1 gesetzt. Lesen des Datenregisters automatisch auf Null.
6	Statusbit für Steuerleitung 2; wie Bit 7
5	Bestimmt Einsatz der Steuerleitung 2 für Eingabe (0) oder Ausgabe (1)
4	Bestimmt Steuersignalart auf Leitung 2
3	Wenn Steuerleitung 2 auf Eingabe, wird hier mit 1 ein Interrupt von Bit 6 möglich. Hilft bei Ausgabe, Signalart festzustellen
2	Bestimmt, ob Daten- (1) oder Datenrichtungsregister (0) angesprochen werden
1	Bestimmt Steuersignalart auf Leitung 1
0	1 ermöglicht hier Interrupts von Bit 7

Der erste Teil unseres PIA-Beispielprogramms versetzt einen 6820-Chip in die Lage, einen Drucker über eine Standard-Centronic-Schnittstelle zu steuern, während der zweite Teil die Steuer- und Datenleitungen festlegt. Dabei signalisiert eine Steuerleitung („Strobe") dem Drucker, daß ein Zeichen „auf dem Weg ist". Der Strobe muß an die Steuerleitung 2 angeschlossen sein, die auf Ausgabe gestellt ist. Ein Steuersignal des Druckers (genannt „Acknowledge" – Bestätigung) zeigt an, daß er bereit ist, das nächste Zeichen zu empfangen. Dieses Signal sollte auf der Steuerleitung 1 ankommen. Die acht Datenleitungen werden dazu mit den acht Datenausgängen des PIA verbunden.

Für die Einstellung der PIA auf diese Aufgabe müssen wir das Datenrichtungsregister anwählen und alle acht Bits auf Ausgabe stellen, dann das Datenregister ansprechen und die Steuerleitung 2 ebenfalls auf Ausgabe stellen. Bei der Ausführung wird das Steuer-/Statusregister so lange gelesen, bis eine Eins in Bit 7 anzeigt, daß der Drucker empfangsbereit ist. Wir schreiben dann ein Zeichen in das Datenregister, das automatisch über die Steuerleitung 2 ein Steuersignal aussendet. Nach der Übertragung des Zeichens wird Bit 6 automatisch auf Eins gesetzt. Das Lesen des Datenregisters löscht nun Bit 6 und 7. Dieser Ablauf wird so lange wiederholt, bis das letzte Zeichen übermittelt ist.

Nehmen wir an, die Basisadresse der PIA liegt in einer Adreßtabelle bei $3000. Beim Aufruf der Druckroutine enthält das Prozessorregister A den Index zu dieser Tabelle und Y die Adresse des Strings, der gedruckt werden soll. Der String hat Normalformat, das heißt, sein erstes Byte ist das Längenbyte. Die erste Subroutine programmiert die PIA, die zweite druckt den String.

Der 6850-ACIA ist ein UART (Universal Asynchronous Receiver/Transmitter), der normalerweise für das RS232-Protokoll und für Modems eingesetzt wird. Er besitzt vier Register und belegt zwei Adressen. Auf der Peripherieseite des Chips liegen fünf Leitungen: eine Leitung für die übermittelten Daten, eine für die empfangenen Daten und drei Leitungen für den Handshaking-Betrieb. Zwei der Steuerleitungen empfangen eingehende Steuersignale – DCD (Data Carrier Detect) und CTS (Clear To Send), während die dritte das ausgehende Signal RTS (Request To Send) sendet. Diese Leitungen werden mit gleichnamigen Leitungen der Standard-RS232-Schnittstelle verbunden.

Die links am Rand stehende Tabelle zeigt die vier ACIA-Register. Mit Bit 7 des Steuerregisters werden Interrupts der empfangenen Daten ermöglicht. Bit 5 und 6 stellen die Übertragungsinterrupts an oder ab und bestimmen die Art des Steuersignals, das auf der RTS-Leitung ausgesandt wird. Bit 2, 3 und 4 bestimmen die Größe des übermittelten „Pakets". Für die serielle Übertragung eines Bytes werden normalerweise mindestens zehn Bits gesandt. Das Anfangsbit zeigt dem Empfänger an, daß Daten folgen. Die Daten selbst können aus sieben oder acht Bits bestehen. An die Daten kann ein Parity-Bit angehängt sein, das zur Feststellung von Übertragungsfehlern dient. Schließlich kommen ein oder zwei Stop-Bits. Insgesamt gibt es folgende Kombinationen:

Register	Offset
Steuerregister	0
Statusregister	0
Daten senden	1
Daten empfangen	2

Steuerregister			Anzahl der Daten-Bits	Parität	Anzahl der Stop-Bits
Bit 4	Bit 3	Bit 2			
0	0	0	7	gerade	2
0	0	1	7	ungerade	2
0	1	0	7	gerade	1
0	1	1	7	ungerade	1
1	0	0	8	keine	2
1	0	1	8	keine	1
1	1	0	8	gerade	1
1	1	1	8	ungerade	1

Die beiden niederwertigen Bits (0 und 1) setzen einen Teiler für die Taktfrequenz und bestimmen damit die Empfangs- und Übertragungsgeschwindigkeit. Da der 6850 keinen eigenen Taktgeber besitzt, wird ein externer verwandt, der normalerweise mit einer Frequenz von 1,760 Hz arbeitet.

Steuerregister		Teiler der Taktfrequenz
Bit 1	Bit 0	
0	0	1
0	1	16
1	0	64

Wenn beide Bits auf Eins stehen, wird im Chip ein „Master Reset" veranlaßt.

Die Bits des Statusregisters haben folgende Funktionen und werden gesetzt, wenn:

Bit	Funktion
7	Interruptanforderung vorliegt
6	bei Übertragung ein Paritätsfehler auftritt
5	Empfänger einen Überlauf hat (es wurden zu viele Bits empfangen, die die zuvor gesandten überschreiben)
4	beim Empfang Fehler in den Rahmenbits (die falsche Anzahl Anfangs- und Endbits) auftritt
3	Signal auf der CTS-Leitung empfangen
2	Signal auf der DCD-Leitung empfangen
1	Übertragungsdatenregister leer
0	Empfangsdatenregister leer

Unser zweites Programmbeispiel empfängt mit dem 6850 eine Zeichenkette (von einem Terminal), die mit Return abgeschlossen ist. Hier wird zunächst der Chip programmiert und dann mit einer Schleife abgefragt, ob das Empfangsdatenregister voll ist. Ist dies der Fall, wird das Datenbyte gespeichert und damit Bit 0 des Statusregisters auf Null gesetzt. Dieser Vorgang wird so lange wiederholt, bis ein Return (ASCII-Code 13) kommt. Das Programm ignoriert Übertragungsfehler, die sich jedoch leicht abfangen lassen, indem man den Inhalt des Statusregisters maskiert und testet, ob eins der Fehlerbits gesetzt ist. Wir setzen dabei ein weit verbreitetes Protokoll voraus: acht Datenbits, keine Parität, zwei Endbits und eine durch 16 geteilte Taktfrequenz. Die erste Subroutine programmiert den Chip, während die zweite die Daten empfängt.

PIA-Programm

```
TABLE   EQU   $3000
        ORG   $1000

        ASLA                    A links drehen, um mit Zwei zu multiplizieren (die Tabelle enthält Zwei-Byte-Adressen)
        LDX   #TABLE
        LDX   A,X               PIA-Adresse holen
        CLR   1,X               Zugang zum Datenrichtungsregister verschaffen
        LDB   #%11111111
        STB   ,X                Alle Bits auf Eingabe setzen
        LDB   #%00101100
        STB   1,X               Interrupts abschalten, Steuerleitung 2 auf Ausgabe setzen und Datenregister anwählen
        RTS

                                Subroutine für das Drucken eines Strings, dessen Adresse in Y steht
        ASLA                    A links drehen, um mit Zwei zu multiplizieren
        LDX   #TABLE
        LDX   A,X               Basisadresse der PIA holen
        LDA   ,Y+               Stringlänge in A speichern
LOOP1   BEQ   FINISH            Testen auf Länge Null
LOOP2   LDB   1,X               Bereit für nächstes Zeichen?
        ANDB  #%10000000        Alle Bits außer Bit 7 maskieren
        BEQ   LOOP2             Falls nicht bereit
        LDB   ,Y+               Nächstes Zeichen holen
        STB   ,X                Drucken
LOOP3   LDB   1,X               Testen, ob Zeichen übertragen
        ANDB  #%01000000        Bit 6 überprüfen
        BEQ   LOOP3             Schleife starten, falls nicht bereit
        LDB   ,X                Datenregister lesen, um die Statusbits zu löschen
        DECA                    Von der Länge Eins abziehen
        BRA   LOOP1             Nächstes Zeichen holen
FINISH  RTS
```

ACIA-Programm

```
TABLE   EQU   $3000             Subroutine für die Programmierung des 6850
        ORG   $1000             Subroutine zur Einstellung des ACIA

ACIAST  ASLA                    A links drehen, um mit Zwei zu multiplizieren (die Tabelle enthält Zwei-Byte-Adressen)
        LDX   #TABLE
        LDX   A,X               Basisadresse des ACIA holen
        LDA   #%00000011
        STA   ,X                Master Reset des ACIA
        LDA   #%00010001
        STA   ,X                Steuerregister ansprechen
        RTS                     ACIA programmieren (8 Datenbits, keine Parität, 2 Stopbits)

                                Subroutine zum Aufnehmen eines Zeichenstrings
BUFFER  EQU   $4000             Hier wird der String gespeichert
CR      EQU   13                ASCII-Code für Return
        BSR   ACIAST            ACIA einstellen, das X-Register enthält die ACIA-Adresse
        LDY   #BUFFER           Bestimmungsort in Y
LOOP    LDB   ,X                Status holen
        ASLB                    Bit 7 aus dem B-Register ins Übertragungsflag des CCR schieben
        BCC   LOOP              Rückverzweigen, falls das Bit nicht gesetzt ist (keine Interruptanforderung)
        LDA   1,X
        STA   ,Y+               Datenbyte holen
        CMPA  #CR               Datenbyte speichern
        BNE   LOOP              Ist A ein Return?
        RTS                     Nächstes Zeichen
```

Mit System unterbrochen

Interrupts sind Meldungen, die den Prozessor bei seiner derzeitigen Aufgabe unterbrechen und ihm wichtige Meldungen zukommen lassen. In diesem Artikel erläutern wir ihren Mechanismus.

Beim Auftreten eines Interrupts beendet der Prozessor den laufenden Befehl und schiebt den aktuellen Inhalt des Befehlszählers auf den Stack. Die Vektoradresse des ausgelösten Interrupts wird nun in den Befehlszähler geladen und so die Steuerung an diese Adresse übergeben (normalerweise eine ROM-Adresse). Ein JMP-Befehl verursacht dann den Sprung auf die eigentliche Interruptroutine. Interrupts werden mit einem RTI-Befehl beendet, der die Steuerung über die im Stack abgelegte Rücksprungadresse an das Hauptprogramm zurückgibt. Da sich der JMP-Befehl im RAM befindet, kann er vom Programmierer abgeändert werden, so daß die Steuerung zunächst auf eine Spezialroutine und dann erst auf die normale Interruptroutine läuft.

Ein umfassender Einsatzbereich für Interrupts sind Tastatureingaben. Wenn ein Programm direkt auf die Tastatur zugreift, der Prozessor aber gerade mit anderen Aufgaben beschäftigt ist, sind keine Eingaben möglich. Selbst während der Bearbeitung von Eingaben können Zeichen verlorengehen.

Die Lösung ist einfach: Beim Drücken einer Taste wird der Prozessor bei seiner Aufgabe unterbrochen und eine Interruptroutine ausgeführt. Diese Routine nimmt das eingegebene Zeichen an und speichert es im Tastaturbuffer. Nach diesem „Umweg" macht der Prozessor mit seiner ursprünglichen Aufgabe weiter.

Nun spricht die Eingaberoutine des Betriebssystems die Tastatur nicht direkt an, sondern nimmt das nächste Zeichen aus dem Buffer (oder wartet, bis dort ein Zeichen gespeichert wird). Dieser Mechanismus stellt sicher, daß auch bei schnellem Tippen keine Zeichen verlorengehen.

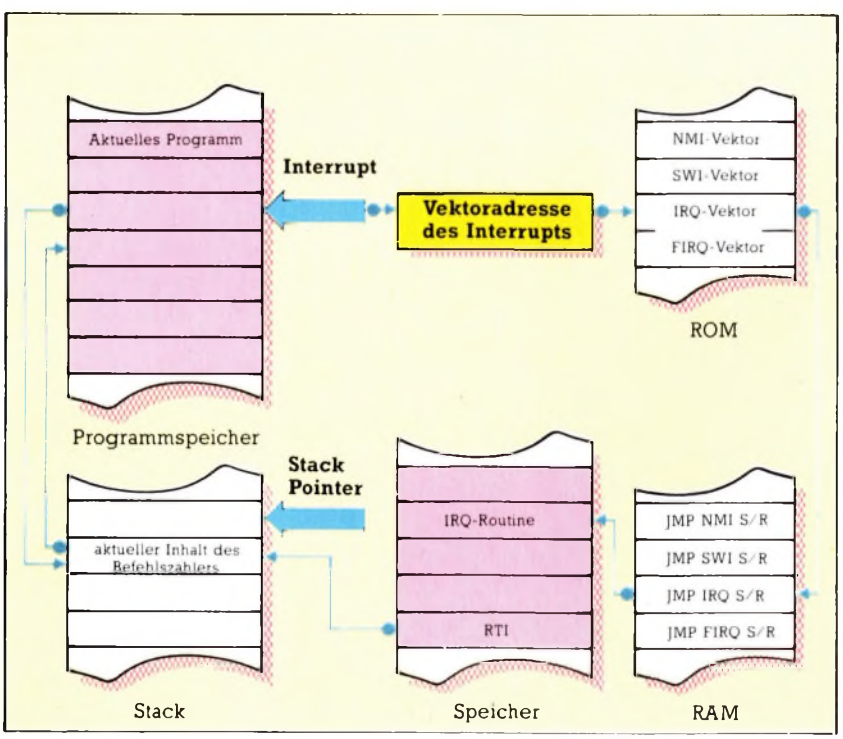
Wenn Eingaben allzu schnell erfolgen, fließt der Buffer über, bevor sie vom Programm bearbeitet werden. Hier hilft nur ein umfangreicher Buffer, der andererseits aber nicht zuviel wertvollen Speicherplatz belegen darf. Ein zweites Problem liegt beim Anwender, der erwartet, daß das eingegebene Zeichen sofort auf dem Bildschirm erscheint.

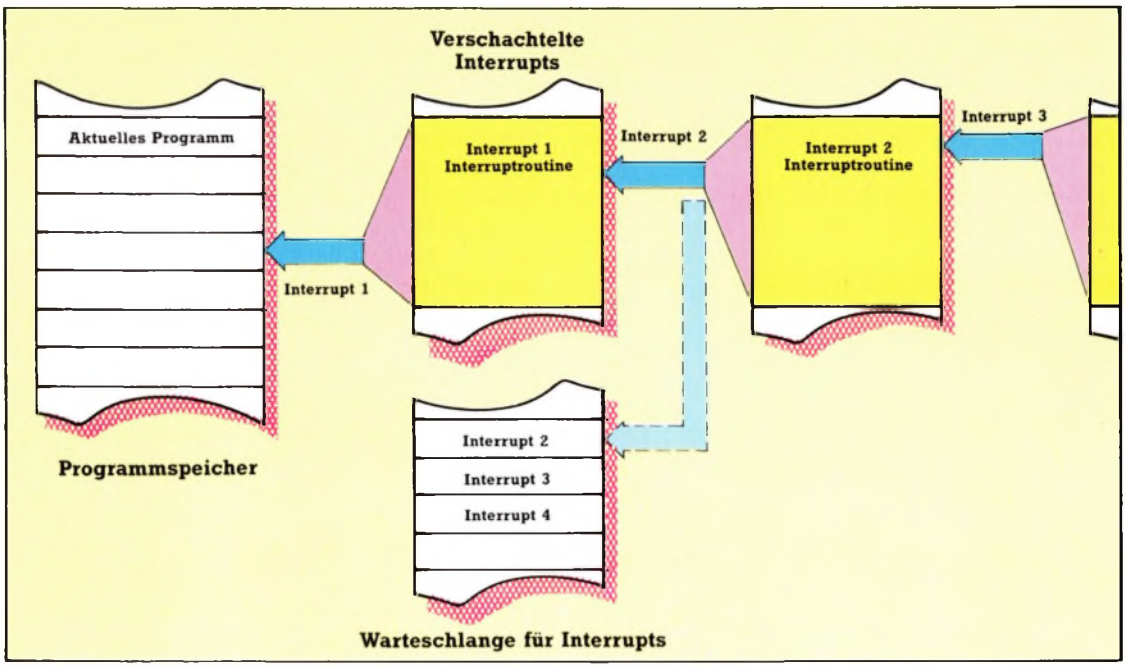
Druckerausgaben sind ein weiteres wichtiges Einsatzgebiet für Interrupts. Bei diesem zeitintensiven Vorgang arbeiten die Prozessoren oft nur 100 Millisekunden (in denen sie Zeichen zum Drucker senden) und warten dann lange Zeit, bis der Drucker die Zeichen verarbeitet hat. Hier bringt ein „Spooler" die Lösung: Alle Dateien, die gedruckt werden sollen, werden in einer Warteschlange zwischengespeichert, und ein Teil der ersten Datei wird in einen kleineren Speicherbuffer geladen. Sobald der Drucker bereit ist, ein weiteres Zeichen zu empfangen, sendet das Interface ein Signal an den Prozessor. Die Interruptroutine übermittelt das nächste im Buffer gespeicherte Zeichen oder holt sich den nächsten Teil aus der Warteschlange. Auf diese Weise läuft der Druckvorgang im Hintergrund, während der Prozessor für andere Aufgaben frei ist.

Bei einigen Abläufen – beispielsweise Diskettenzugriffen – kann eine Unterbrechung zu Datenverlusten oder anderen fatalen Fehlern führen. Durch die „Maskierung" der Interrupts hat der Prozessor jedoch die Möglichkeit, Unterbrechungen während besonders wichtiger Abläufe zu ignorieren. Hier zeigt oft eine Meldung das Auftreten eines Interrupts an, der dann später behandelt wird.

Im umgekehrten Fall – wie etwa bei interruptgesteuerten Diskettenschnittstellen – sind Interrupts nicht maskierbar und haben immer Priorität. Nicht maskierbare Interrupts können aber auch von Schaltungen ausgelöst werden, die ein Abfallen der Stromversorgung erkennen. Solange noch genügend Strom vorhanden ist, speichert eine Spezialroutine sofort die aktuelle Aufgabe.

Wenn mehrere Quellen Unterbrechungen auslösen können, muß man auch die Möglichkeit „verschachtelter" Interrupts in Betracht ziehen. Ein Interrupt, das während der Behandlung eines anderen Interrupts auftritt, läßt sich auf zwei verschiedene Arten abfangen. Zunächst kann das neue Interrupt so lange ignoriert werden, bis das laufende beendet ist.





Wenn während eines Interrupts eine neue Unterbrechung auftritt, können Interrupts verschachtelt werden. Dabei wird beim Auftreten eines Interrupts der Befehlszähler auf den Stack geschoben, der neue Interrupt ausgeführt und dann die Steuerung an die gespeicherte Adresse zurückgegeben. Verschachtelungen dieser Art sind natürlich von der Kapazität des Stacks abhängig und auch von der Fähigkeit der Interruptroutinen, Verzögerungen abfangen zu können.

Der Prozessor kann aber auch die Einzelheiten jedes Interrupts in einer speziellen Warteschlange speichern. Sobald das erste Interrupt abgeschlossen ist, untersucht der Prozessor diese Schlange und führt jedes dort gespeicherte Interrupt nacheinander aus, bis er schließlich die Steuerung wieder an das Hauptprogramm zurückgibt.

Interrupts lassen sich aber auch mit unterschiedlichen Dringlichkeitsstufen versehen, damit höhere Prioritäten niedrigere Prioritäten unterdrücken können. Das funktioniert aber nur, wenn das Betriebssystem verschachtelte Interrupts zuläßt.

Der SWI-Befehl ist eine bequeme Möglichkeit, über die Erzeugung eines eigenen Interrupts vom Programm in das Betriebssystem zurückzukehren. Diese Methode heißt „Software Interrupt" (zur Unterscheidung zu den bisher untersuchten „Hardware Interrupts"). SWI-Befehle lassen sich auch als Unterbrechungspunkte in Maschinencodeprogramme einsetzen und erleichtern so die Fehlersuche.

Registerinhalte untersuchen

Der Anwender bestimmt, an welchen Punkten die Programmausführung eine Pause machen soll. Die an diesen Punkten stehenden Befehle werden gegen SWIs ausgetauscht. Beim Ablauf des Programms erlaubt die entsprechende Interruptroutine dem Programmierer dann, Registerinhalte zu untersuchen und zu verändern. Damit läßt sich exakt feststellen, welche Auswirkung einzelne Befehle haben. Bei der Wiederaufnahme der Programmausführung setzt der Monitor (oder das Fehlersuchprogramm) die am Unterbrechungspunkt dargestellten Befehle wieder ein und läßt das Programm weiterlaufen.

Der 6809 besitzt drei unterschiedliche Interruptmechanismen: IRQ (Interrupt Request – einfache Interruptanforderung), FIRQ (First Interrupt Request – schnelle Interruptanforderung) und NMI (Non-Maskable Interrupt – nicht maskierbarer Interrupt). Alle drei werden durch den Empfang des entsprechenden Signals auf drei Kontakten des Prozessorchips aktiviert. Der Balken über dem Namen (beispielsweise in $\overline{IRQ}$) zeigt an, daß sie beim Prozessor von einem Nullsignal und nicht durch eine Eins aktiviert werden. Diese drei Kontakte sind mit dem Hauptbus verbunden, damit Peripheriechips wie der 6820 und 6850 ihre Ausgabekontakte auf die gleichen Leitungen legen können. Durch die Programmierung der Chips können die Interrupts ermöglicht werden, so daß die entsprechenden Signale automatisch gesandt werden.

Weiterhin gibt es drei Softwareinterrupts mit den Codes SWI1, SWI2 und SWI3.

Beim Auftreten eines Interrupts wird die Steuerung an eine Vektoradresse übergeben, die in einer speziellen Speicherstelle an der oberen Speichergrenze liegt. Diese Vektoradressen befinden sich normalerweise im ROM, so daß die Steuerung immer an die gleiche feste Adresse übergeben wird, die jedoch im RAM liegt. Sie enthält einen Sprungbefehl, mit dem der Anwender eine eigene Routine ansprechen kann. Die Speicherstellen sind:

Interrupttyp	Vektor
$\overline{NMI}$	$FFFC
SWI	$FFFA
$\overline{IRQ}$	$FFF8
$\overline{FIRQ}$	$FFF6
SW12	$FFF4
SW13	$FFF2

Interessant ist hierbei, daß die beiden höchsten Speicherbytes $FFFE und $FFFF den „Reset Vektor" enthalten, also die Adresse, an die die Steuerung beim Einschalten des Systems oder bei einem Hardware-Reset übergeben wird. Im Normalfall liegt dort die Anfangsadresse des ROM-Monitors.

Drei Bits des Condition Code Registers liefern Informationen über die Interrupts: Bit 4 (I),

Bit 6 (F) und Bit 7 (E). Das Setzen von Bit I maskiert den $\overline{IRQ}$ Interrupt und Bit F den $\overline{FIRQ}$. Mit dem E-Bit kann der Prozessor zwischen $\overline{IRQ}$, $\overline{NMI}$ und $\overline{FIRQ}$ unterscheiden. Steht es auf Eins, ist ein $\overline{IRQ}$ oder ein $\overline{NMI}$ eingetreten, bei Null ein $\overline{FIRQ}$.

Ein Interrupt wird wie ein Subroutinenaufruf behandelt: Der Inhalt einiger oder aller Register wird auf den Stack geschoben, so daß die Steuerung an die gleiche Stelle zurückgegeben werden kann, an der das Programm unterbrochen wurde. Ähnlich wie RTS wird die Interruptroutine mit einem RTI-Befehl beendet, der die Register vom Stack zieht und die Steuerung an das Programm zurückgibt.

Der Unterschied zwischen $\overline{FIRQ}$ und den anderen Interrupts liegt darin, daß $\overline{FIRQ}$ nur den Befehlszähler und das Condition Code Register auf den Stack schiebt und daher weitaus schneller ist. Da die Interruptroutinen alle eingesetzten Register wieder in den ursprünglichen Zustand versetzen müssen, sollte dieser Interrupttyp nicht mit Routinen eingesetzt werden, die mehr als zwei Register benötigen. Hier wird auch die Aufgabe des E-Bits deutlich: Da RTI beide Routinen – $\overline{IRQ}$ und $\overline{FIRQ}$ – beendet, kann der Prozessor mit dem E-Bit feststellen, welche Register vom Stack gezogen werden müssen. Ein Interrupt läuft folgendermaßen ab:

1) Der laufende Befehl wird ausgeführt.
2) Das I-Bit wird gesetzt, und alle $\overline{IRQ}$-Interrupts werden abgeschaltet. Bei einem $\overline{FIRQ}$ oder $\overline{NMI}$ wird weiterhin Bit F gesetzt, um auch $\overline{FIRQ}$ abzuschalten. SWI1 kann andere Interrupts maskieren, SWI2 und SWI3 jedoch nicht.
3) Bei $\overline{FIRQ}$ wird Bit E auf Null gesetzt, in allen anderen Fällen auf Eins.
4) Der Vektor in der entsprechenden Speicherstelle wird in den Befehlszähler geladen und die Steuerung an diese Adresse übergeben.

Unser erstes Programm zeigt ein weiteres Einsatzgebiet für Interrupts: die Steuerung einer Echtzeituhr. Wir setzen voraus, daß ein Zeitmechanismus – beispielsweise ein Spezialchip wie der 6840 Interval Timer, eine Unterteilung der Systemuhr oder eine Modifikation der 50-Hz-Stromversorgung – bei $5000 mit der PIA verbunden ist. Die erste Subroutine ermöglicht die Interrupts und richtet bei $50 einen 16-Bit-Zähler ein. Die Interruptroutine inkrementiert diesen Zähler, so daß wir bei jeder Untersuchung von $50 die Anzahl der Zeitsignale erhalten, die von einem bestimmten Zeitpunkt an empfangen wurden. Aus diesen Informationen und der Frequenz der Impulse kann die aktuelle Zeit berechnet werden.

Unser zweites Programm nimmt an, daß bei $E000 ein Drucker mit der gleichen PIA angeschlossen ist. Wir verwenden einen Buffer mit unbegrenzter Kapazität (bei $100), in den die Hilfsroutine eine Ausgabezeile speichert. Während des Druckes wird bei $50 ein Flag auf Null gesetzt und nach Ende des Druckes auf Eins. Dadurch weiß eine weitere Routine (auf die wir hier nicht weiter eingehen), wann sie den Buffer füllen soll. $51 und $52 enthalten einen Zeiger für die Adresse des nächsten Bufferzeichens, das gedruckt werden soll. Die erste Subroutine richtet die PIA, das Flag und den Buffer auf eine neue Zeile aus.

Erstes Programm

```
PIACR   EQU   $E001    PIA-Steuerregister
PIADR   EQU   $E000    PIA-Datenregister
INTRP   EQU   $2000
CLK1    EQU   $50
CLK2    EQU   $51
        ORG   $1000    Subroutine zur Initialisierung der Uhr
INITCK  CLR   CLK1     Uhrspeicher loschen
        CLR   CLK2
        LDA   #%00000101   PIA-Interrupts ermoglichen
        STA   PIACR
        ANDCC #%11101111   IRQ ermoglichen
WAITCK  TST   CLK2     Auf erste Inkrementierung warten
        BEQ   WAITCK
        RTS
        ORG   INTRP    Interruptroutine
        LDA   PIADA    Interrupt loschen
        LDD   CLK1     Zahler holen
        ADDD  #1       Zahler inkrementieren
        STD   CLK1
        RT1
```

Zweites Programm

```
PIACR   EQU   $E001    PIA-Steuerregister
PIADR   EQU   $E000    PIA-Datenregister
INTRP   EQU   $2000
CR      EQU   13       Wagenrucklauf
BUFFER  EQU   $100     Bufferadresse
BUFPTR  EQU   $51      Bufferzeiger
FLAG    EQU   $50      Flag für Zeilenende
        ORG   $1000    Subroutine für Initialisierung
        CLR   FLAG     Flag für Zeilenende loschen
        LDX   #BUFFER  Zum Anlegen des Buffers Bufferzeiger
                       initialisieren
        STX   BUFPTR
        CLR   PIACR    Adreßdaten-Richtungsregister
        LDA   $FF      Alle Leitungen auf Ausgabe stellen
        STA   PIADR
        LDA   #%00000101   PIA-Interrupts ermoglichen
        STA   PIACR
        ANDCC #%11101111   IRQ ermoglichen
        RTS
        ORG   INTRP    Interruptroutine
        LDX   BUFPTR   Bufferzeiger
        LDA   ,X+      Nächstes Zeichen aus Buffer holen
        STA   PIADR    Zeichen drucken
        LDB   PIADR    Interrupt löschen
        STX   BUFPTR   Bufferzeiger inkrementieren
        CMPA  #CR      Ist dies das Zeilenende?
        BNE   FINISH   Überspringen falls nicht
        INC   FLAG     Sonst Flag setzen
FINISH  RTI
```

Rangiermethoden

Wir gehen auf den verschiebbaren (relocatable) Code, die Befehlslängen und verschiedene Ausführungsgeschwindigkeiten ein.

Ein Programm mit verschiebbarem oder positionsunabhängigem Code kann in jeden beliebigen Speicherbereich geladen werden und dort ohne Änderungen funktionieren. Besonders bei Mehrplatzsystemen und Hintergrundprozessen hat dies große Bedeutung, da dabei mehrere Programme gleichzeitig in den Speicher geladen werden können und das Betriebssystem den Code oft an verschiedenen Positionen unterbringen muß. Doch selbst bei Einplatzsystemen sollten Unterroutinen als selbständige Module angelegt sein, da sie sich dann ohne Änderungen auch in anderen Programmen einsetzen lassen.

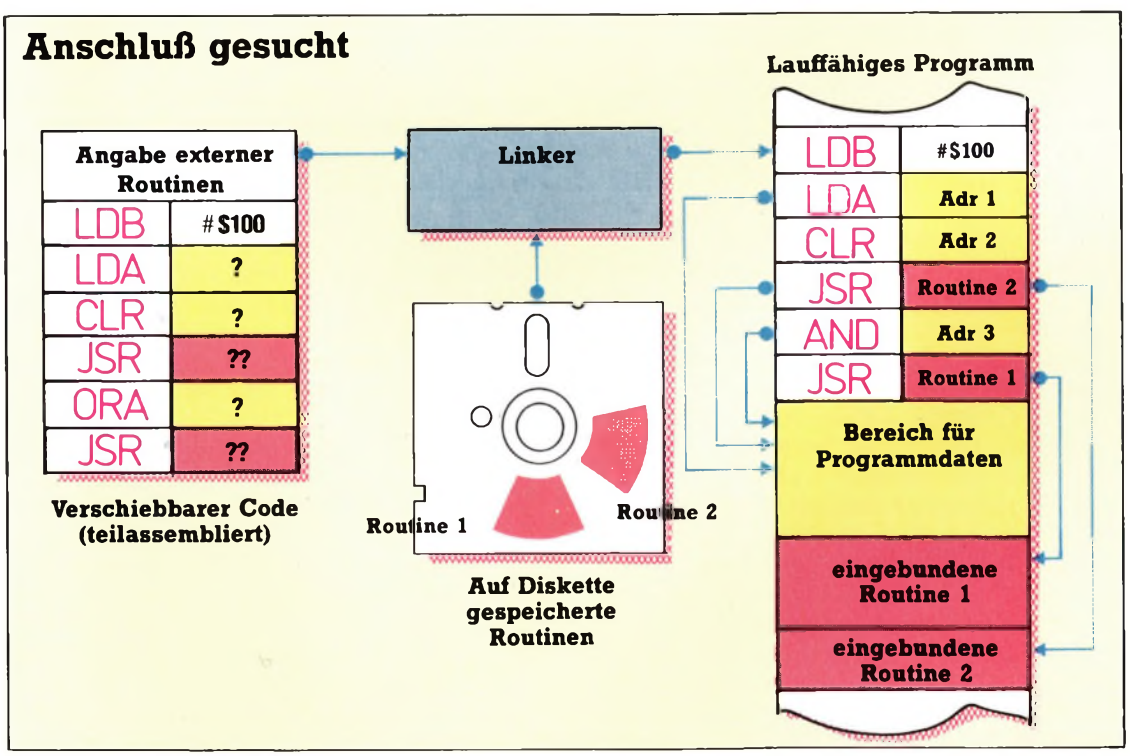
Die meisten Prozessoren verwenden einen sogenannten „Linker", um Adressen in den vom Assembler erstellten verschiebbaren Code (der nur relative Speicherstellen angibt) einzusetzen. Der Linker stellt dabei sicher, daß die Steuerung zwischen den einzelnen Modulen korrekt übergeben wird. Auf diese Weise ist es sogar möglich, die Codes unterschiedlicher Sprachen in den gleichen verschiebbaren Code zu compilieren oder zu assemblieren. PASCAL-Programme können so beispielsweise mit FORTRAN-Modulen arbeiten. Diese Technik ist auch auf dem 6809 möglich. Der 6809 bietet zusätzlich den Vorteil, daß ein verschiebbarer Code direkt eingegeben werden kann und die Adressen nicht erst durch einen zusätzlichen und zeitraubenden Arbeitsgang erstellt werden müssen.

Der Schlüssel zum verschiebbaren Code ist das „Offset", das alle Adressen mit Bezug auf den Befehlszähler (PC) angibt. Programme setzen Adressen auf zwei Arten ein: als Daten oder als Adressen, an die die Steuerung übergeben wird. Verzweigungsbefehle wie BRA oder BSR berechnen ihre Sprungadressen als Offsets vom PC und sollten daher für alle Sprünge innerhalb eines Anwenderprogramms verwandt werden. Absolute Sprungbefehle wie JMP und JSR sollten nur Adressen ansprechen, die sich im Speicher immer auf der gleichen Position befinden, beispielsweise die Routinen des Betriebssystems.

Weitaus schwieriger ist der unabhängige Bezug auf Datenpositionen. Der 6809 kann dafür den PC als Index einsetzen. Der Befehl

```
LDA OFFSET,PC
```

addiert das Offset (mit Vorzeichen) zu dem aktuellen Wert des PC und erhält so die „effektive Adresse". Allerdings ist diese Adressierungsart nicht ganz einfach, da bei der Berechnung der Differenz zwischen Datenadresse und dem Wert des PC berücksichtigt werden



Bei größeren Systemen werden die Maschinencodeprogramme vom „Linker" in den Speicher geladen. Dieses Dienstprogramm des Betriebssystems übernimmt vom Assembler den teilassemblierten Code (der keine absoluten Adressen enthält) und bestimmt aus dem aktuellen Status des Systems die geeignetste ORG-Adresse. Mit dieser „Schlüsseladresse" tauscht der Linker die vom Assembler eingesetzten symbolischen Programmadressen gegen absolute Adressen aus und bindet die vom Programmierer angegebenen externen Module in den Code ein. Die absoluten (CALL) Adressen der externen Module werden dabei gegen die symbolischen Adressen des Programms ausgetauscht. Schließlich übergibt der Linker das vollständige Programm zur Ausführung an das Betriebssystem.

muß, daß der PC unmittelbar nach dem Laden eines Befehls inkrementiert wird. Nach der Ausführung eines Befehls zeigt der PC daher immer auf die darauffolgende Instruktion.

Die vielen möglichen Befehlslängen des 6809 (ein bis fünf Byte) verkomplizieren diese Methode. So belegt

```
LDX OFFSET,Y
```

beispielsweise ein Byte für den Op-Code und ein Byte für das nachgestellte Byte, das bei jedem Indexvorgang angibt, welches Indexregister eingesetzt wird und ob die Adressierung indirekt erfolgen soll. Das Offset kann Null, ein oder zwei Byte belegen. Außerdem können Null-Offsets und Offsets, die sich in fünf Bits ausdrücken lassen, in das nachgestellte Byte integriert werden (einige Assembler haben damit jedoch Probleme). Längere Offsets benötigen entweder ein zusätzliches Byte (bis acht Bits Länge) oder zwei Byte. Spezielle Offsets für Null oder im Fünf-Bit-Format sind jedoch nicht möglich, wenn der PC als Index eingesetzt wird.

```
LDY OFFSET,X
```

würde ein zusätzliches Byte belegen, da der Op-Code für LDY zwei Byte lang ist.

Wenn Sie gern in der Assemblersprache programmieren und mit Datenadressen und Subroutinen umgehen können, wird Ihnen die Berechnung der Op-Code-Längen und der Offsets schon bald zur Selbstverständlichkeit werden. Sie können sich aber auch einen Assembler kaufen, der Ihnen diese Arbeit abnimmt. Die meisten Assembler verfügen über die Anweisung PCR (Programm Counter Relative – relativer Bezug zum Befehlszähler), die den Assembler anweist, den PC als Indexregister einzusetzen und den Offset selbst zu berechnen. Hier ein Beispiel:

```
DATITM FCB 0
.............
        LDA DATITM,PCR
```

Escape-Sequenzen

In unserem Programmbeispiel setzen wir die Offset-Technik ein, um Terminals zu emulieren (imitieren). Damit funktionieren Programme, die auf ein bestimmtes Terminal ausgerichtet sind, auch auf anderen Geräten. Die Unterschiede zwischen einzelnen Terminals zeigen sich besonders in den Codes, die die Bildschirmfunktionen (die Cursorpositionierung oder das Löschen des Schirms) steuern. Dafür werden entweder Steuercodes (Zeichen mit einem ASCII-Code unter 32) oder „Escape-Sequenzen" eingesetzt, die aus dem Escape-Zeichen (ASCII 27) gefolgt von einem oder mehreren Zeichen bestehen. Unser Programm ist recht einfach und kann nur jeweils ein Steuerzeichen durch ein anderes oder eine Escape-Sequenz durch eine andere ersetzen. Es zeigt aber, wie eine Emulation funktioniert. Die Routine legt zwei Tabellen an: eine für Steuerzeichen und die zweite für Escape-Sequenzen. Das Originalzeichen dient dabei als Tabellen-Offset für das Austauschzeichen.

Der Code ist frei verschiebbar und kann problemlos an jedes Programm angehängt werden. Wir setzen voraus, daß die Betriebssystemroutine OUTCH das im Akkumulator A gespeicherte Zeichen an den Bildschirm sendet. Da wir den Befehl JMP einsetzen, sollte sich diese Routine an einer festen Speicheradresse befinden. Beachten Sie, daß die ORG-Anweisung gegeben werden muß, obwohl sie hier keine direkte Funktion hat. Das angezeigte Zeichen sollte ein A sein.

Vier Regeln

Nicht nur bei der PCR-Adressierung müssen Befehlslängen errechnet werden. Oft wird die Gesamtlänge einer Routine gebraucht, damit sie in einen begrenzten Speicherbereich eingepaßt werden kann. Fast jedes Buch über den 6809-Assembler enthält eine Tabelle der mnemotischen Befehle und ihrer Daten. Bei jedem mnemotischen Kürzel steht der zugehörige Op-Code, die Gesamtlänge des Befehls, die Zahl der Taktzyklen, die der Befehl zur Ausführung benötigt, und die Auswirkungen auf die Flags des Condition Code Registers.

Es gibt vier allgemeine Regeln für die Berechnung von Befehlslängen:

1) Die meisten Op-Codes sind ein Byte lang. Op-Codes, die den Inhalt von S und Y ansprechen (außer LEA) und einige, die U einsetzen (LDY und STS), haben dagegen die Länge von zwei Byte.

2) Jede indizierte Adressierung braucht ein nachgestelltes Byte (Post-Byte) und möglicherweise je nach Länge des Offsets weitere ein oder zwei Byte.

3) Daten, die den Op-Codes für unmittelbare Adressierung folgen, sind je nach eingesetztem Register ein oder zwei Byte lang.

4) Direkte Adressen sind ein Byte lang, alle anderen zwei Byte. Nicht alle Assembler verwenden die direkte Adressierung optimal, so daß Adressen oft zwei Byte haben, die nur ein Byte brauchen.

Die vielen möglichen Befehlslängen verkomplizieren auch die Berechnung der Ausführungszeit: Sie ist von den Byte abhängig, die geladen werden müssen. Diese Information ist besonders für Echtzeitanwendungen und einige Peripherietreiber wichtig. Da die Ausführungszeit in Taktzyklen angegeben wird – oder zumindest in der geringstmöglichen Anzahl Taktzyklen –, hängt die reale Ablaufzeit von der Taktfrequenz ab. Auf dem 6809 wird oft ein MHz eingesetzt – eine Million

Zyklen pro Sekunde. Damit dauert jeder Taktzyklus eine Millionstelsekunde. Der Befehl

```
LDA DATITM
```

verwendet eine 16-Bit-Adresse und braucht fünf Zyklen – er dauert daher fünf Millionstelsekunden.

```
PSHS PC, B, CC
```

braucht fünf Zyklen plus einen Zyklus für jedes Byte, das auf den Stack geschoben wird – insgesamt also neun Zyklen, da der PC aus zwei Byte besteht.

Wenn ein System keine Echtzeituhr enthält, kann die Zeit nur durch eine Verzögerungsroutine gemessen werden. Dabei wird eine Befehlsfolge ausgeführt, deren Ablaufzeit die Länge der gewünschten Routine simuliert. Bei einer Taktfrequenz von einem MHz ruft unsere Routine Verzögerungen im Bereich von 1 bis 255 Millisekunden hervor. Die Anzahl der Millisekunden (ms) wird als Parameter an A (den Akkumulator A) übergeben.

Die Berechnung der Konstanten COUNT läßt sich folgendermaßen darstellen:

Befehl	Anzahl der Taktzyklen	Anzahl der Wiederholungen	Benötigte Zeit (Taktzyklen
PSHS B,CC	7	1	7
LDB #COUNT	2	(A)	(A) * 2
DECB	2	(A) * COUNT	(A) * COUNT * 2
BNE LOOP2	3	(A) * COUNT	(A) * COUNT * 3
DECA	2	(A)	(A) * 2
BNE LOOP1	3	(A)	(A) * 3
PULS PC,B,CC	9	1	9

Die Ablaufzeit errechnet sich aus dem Gesamtwert von (A) * (7 + 5 * COUNT) + 16 Taktzyklen. Damit die Berechnung einfacher wird, ignorieren wir die 16 Taktzyklen. (Da Intervalle dieser Art üblicherweise in Millisekunden gemessen werden, sind die 16 Millionstelsekunden unerheblich.) Bei einer Taktfrequenz von einem MHz enthält eine Millisekunde 1000 Taktzyklen. Die Gesamtzahl ist daher (A) * 1000 Taktzyklen.

(A) * (7 + 5 * COUNT) = (A) * 1000
(7 + 5 * COUNT) = 1000
5 * COUNT = 973
COUNT = 195
(nächste Ganzzahl)

Zwar können andere Routinen exaktere Verzögerungszeiten errechnen oder mit 16-Bit-Adressen einen größeren Bereich umfasssen, alle dekrementieren jedoch ein Register um eine feste Zahl.

Emulations-Routine

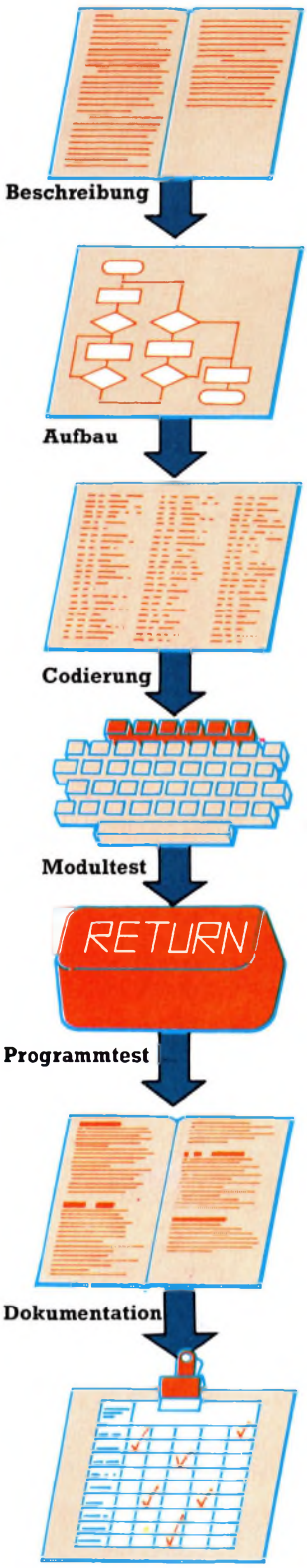
```
ESCAPE  EQU   27
SPACE   EQU   32          (Leerzeichen ist ASCII 32)
OUTCH   EQU               Hier die Betriebssystemadresse eintragen
        ORG   $1000
CTABLE  RMB   32          Tabelle der Steuerzeichen
ETABLE  RMB   128         Tabelle der Escape-Zeichen
EFLAG   FCB   0           Dieses Flag zeigt an, ob das letzte Zeichen ein Escape war
DISPCH  PSHS  X           X speichern
        TST   EFLAG, PCR  Testen, ob das letzte Zeichen ein Escape war
        BEQ   DISP1       Falls kein Escape, auf DISP1 verzweigen
        LEAX  ETABLE, PCR Sonst die Adresse von CTABLE nach X
        LDA   A,X         Mit dem Originalzeichen als Offset das Austauschzeichen in A laden
        CLR   EFLAG,PCR   EFLAG löschen
        BRA   FINISH
DISP1   CMPA  SPACE       Testen, ob Steuerzeichen
        BGE   FINISH      Falls kein Steuerzeichen, zu FINISH verzweigen
        CMPA  ESCAPE      Sonst auf Escape testen
        BEQ   ESCCH       Falls ja, auf ESCCH verzweigen
        LEAX  CTABLE,PCR  Adresse von CTABLE in X laden
        LDA   A,X         Mit dem Originalzeichen als Offset das Austauschzeichen in A laden
        BRA   FINISH
ESCCH   INC   EFLAG,PCR   EFLAG setzen, um anzuzeigen, daß das Zeichen ein Escape war
FINISH  PULS  X           X wiederherstellen
        JMP   OUTCH       Zeichen in A anzeigen
        END               Hier gibt RTS am Ende von OUTCH die Steuerung an das aufrufende Programm zurück
```

Verzögerungs-Routine

```
                          Subroutine für Verzögerungen um (A) Millisekunden
COUNT   EQU   195         Siehe Berechnung
        ORG   $1000
DELAY   PSHS  B,CC        Die anderen beiden eingesetzten Register sichern
LOOP1   LDB   #COUNT      Eine Millisekunde zählen
LOOP2   DECB              Weiter dekrementieren
        BNE   LOOP2       Bis B Null erreicht
        DECA              Jede ms A dekrementieren
        BNE   LOOP1       Bis A Null erreicht
        PULS  PC,B,CC     Zurückspringen
```

Mit Struktur

In dieser Serie wurden zunächst der Befehlssatz des 6809 untersucht und dann einige einfache Routinen angelegt. Nun stellen wir Techniken vor, mit denen sich umfangreiche Assemblerprogramme strukturieren lassen.



Es ist zwar nicht immer einfach, Maschinencode strukturiert zu programmieren, doch macht sich der Aufwand durch kürzere Testzeiten und übersichtliche Programme bezahlt.

Wenn korrekt angelegte Programme, modularer Aufbau und strukturierte Programmierung schon bei höheren Programmiersprachen große Vorteile bringen, so kommt man bei dem komplizierten Code der Maschinensprache ohne Systematik überhaupt nicht mehr zurecht. Nun besitzt die Assemblersprache keine vorgefertigten Strukturen wie WHILE . . .WEND oder IF. . .THEN . . .ELSE, die von sich aus schon eine Systematik vorgeben. Es gibt keine einheitliche Schreibweise, keine Datentypen für Variablen, und im Vergleich mit Hochsprachen haben Assemblerprogramme die sechs- bis zehnfache Menge von Befehlen. Auch Fehler mit weitreichenden Konsequenzen können leicht vorkommen: Ein einziges fehlerhaftes Byte kann durchaus alle Daten einer Diskette löschen. In diesem Artikel stellen wir daher eine Methode vor, die den Umgang mit der Assemblersprache erleichtert.

Schon früh haben Programmierer erkannt, daß Vorrausplanung und Klarheit die Grundlage jedes erfolgreichen Programmierstils ist. Doch obwohl das Konzept der strukturierten Programmierung in der Welt der Microcomputer noch neu und ungewohnt zu sein scheint, arbeiten mehr und mehr Amateure mit den professionellen Programmiertechniken. Wenn Sie an Ihre ersten unstrukturierten, nicht dokumentierten und hand-assemblierten Maschinencodeprogramme zurückdenken, werden Sie diese Entwicklung auch bei sich selbst feststellen können. Durch klaren Programmaufbau und transparente Arbeitsmethoden entstehen auch gute Programme.

- Aufgabenbeschreibung: In diesem Stadium sollten Sie den Ein- und Ausgaben spezielle Aufmerksamkeit widmen. Hier müssen besonders die Signale der direkt angesprochenen Peripheriegeräte (zum Beispiel Tastatur und Bildschirm) berücksichtigt werden. Dabei können Zeitprobleme auftreten oder auch Routinen fehlen (beispielsweise die Umwandlung der vom Programm eingelesenen ASCII-Zeichen in binär codierte Dezimalzahlen). Achten Sie darauf, daß nicht nur die ankommenden Daten definiert werden, sondern auch das Format, in dem das Programm sie weiterverarbeitet.
- Programmaufbau: Im zweiten Stadium entwirft man die Prozesse, die die Eingaben in Ausgaben umwandeln. Routinen und Daten sollten dabei möglichst in logischen Blöcken gruppiert und als selbständige Module angelegt werden. Es gibt zwei Techniken, Programme zu zerlegen: „vom Teil zum Ganzen" – das Programm wird Stück für Stück aus Einzelmodulen aufgebaut, und „vom Ganzen zum Teil" – hier wird das Programm in immer kleinere Einheiten zerlegt. Das Hauptaugenmerk liegt dabei auf der Funktion und nicht auf der Umsetzung in den Programmcode. Erst nach Definition dieser Funktionsmodule wird mit der Assemblierung begonnen.

Vom Teil zum Ganzen

Bei der Methode „vom Teil zum Ganzen" werden fertige Standardmodule verwendet, die sich leicht zusammensetzen lassen und den Speicher besser ausnutzen. Leider werden die Programme dadurch fehleranfällig, schwer zu testen und unübersichtlich. Die zweite Methode führt zu strukturierten Programmen, die sich leicht modulweise testen lassen. Dabei werden die Programmschritte durch Kurzroutinen voneinander getrennt und einzeln überprüft. Die Kurzroutinen ersetzen fehlende Module. Sie nehmen die Eingaben entgegen und liefern ohne Verarbeitung die korrekten Ausgaben. Nach dieser Methode entwickelte Programme belegen meistens jedoch mehr Speicherplatz. Auch lassen sich ihre Module nur selten in andere Programme einsetzen.

Nun werden für jedes einzelne Modul der Datenbedarf, die Datenstrukturen und die Algorithmen definiert. Hier kann ein Ablaufdiagramm die Aufgabe sehr vereinfachen. Viele Programmierer ziehen allerdings eine Art Hochsprache – den „Pseudo-Code" – vor oder arbeiten mit BASIC. Nachdem Algorithmen und Daten in einer vertrauten Sprache entworfen worden sind, brauchen sie nur noch vom Pseudo-Code in die Assemblersprache übersetzt zu werden – die einzige Aufgabe, bei der Sie es direkt mit dem Maschinencode zu tun haben. Diese Methode ist einfacher als eine gleichzeitige Entwicklung und Codierung in der Assemblersprache.

- Codierung: Bei gut strukturierten Routinen ist dieser Schritt der einfachste und schnellste. Die Codierung überträgt die Steuerstrukturen der Hochsprache auf die Maschinenebene. BRA und JMP sollten nicht wahllos eingesetzt werden, da ein unübersichtlicher Code später die

Fehlersuche sehr erschwert. Unser Diagramm zeigt, wie die bekanntesten Steuermechanismen programmiert werden. Der Einfachheit halber setzen wir voraus, daß alle Daten ein Acht-Bit-Format haben.

Der Code für Steuerstrukturen wird oft länger als notwendig. Wenn ausreichend Platz vorhanden ist, sollten Sie sich jedoch nicht weiter darum kümmern: Ein kürzerer Code führt nicht unbedingt zu kürzeren Ablaufzeiten, verursacht aber oft zusätzlichen Aufwand bei der Programmentwicklung und beim Testen. Bei beschränktem Speicherplatz lohnt es sich allerdings, zunächst den ausführlich strukturierten Code zu schreiben und ihn später zu optimieren. Erst zu diesem Zeitpunkt müssen Sie die Speicherplatzgrenzen berücksichtigen.

● Modultest: In diesem Stadium wird jedes Modul einzeln (wenn nötig mit Kurzroutinen) getestet. Damit ist sichergestellt, daß gültige Eingaben auch gültige Ergebnisse erzeugen. Die Fehlersuche der Assemblersprache unterscheidet sich grundlegend von der Fehlersuche in BASIC. Um die Abläufe überhaupt verfolgen zu können, müssen die vom Programm eingesetzten Speicherstellen und Registerinhalte untersucht und (unter Umständen) verändert werden. Es ist kaum möglich, Assemblerprogramme ohne Hilfsroutinen für das Setzen und Löschen von Unterbrechungspunkten fehlerfrei zu bekommen. Das Programm läuft dabei bis zu einem Haltepunkt, an dem die Hilfsroutine den Registerinhalt anzeigt und die Möglichkeit zur Änderung gibt.

● Test des vollständigen Programms: Sobald die Einzelmodule fehlerfrei sind, wird das Programm zusammengesetzt und mit Testdaten geprüft. Wenn alle Einzelmodule fehlerfrei laufen, ist diese Aufgabe recht einfach.

● Dokumentation: Da Assemblerprogramme komplizierter sind als Programme in Hochsprachen, ist eine ausführliche Dokumentation sehr wichtig. Besondere Aufmerksamkeit sollte dabei dem Speichereinsatz, den Stacks und den von Subroutinen eingesetzten Registern gewidmet werden.

Sorgfalt zahlt sich aus

● Wartung: Programme, die längere Zeit gelaufen sind, müssen oft überarbeitet werden. Dabei werden Fehler beseitigt, die während der Programmentwicklung nicht erkannt wurden, oder Verbesserungen eingebaut. Hier zahlt sich eine sorgfältige Programmierung und ausführliche Dokumentation besonders aus: Wenn keine Struktur vorhanden ist und/oder die Dokumentation fehlt, lassen sich Änderungen unter Umständen nicht mehr einfügen, und das Programm muß völlig neu geschrieben werden. Auf die Weise kann man aber kaum sein Talent in Assembler entfalten.

All diese Entwicklungsstufen zeigen wir am Beispiel eines Monitor/Debugger (Programm für Maschinencodeanzeige und Fehlersuche). Wenn Sie zuvor mit einem Assembler gearbeitet haben, sind Sie mit den Funktionen eines Monitor/Debuggers sicher schon vertraut. Das Programm unterstützt den Assemblerprogrammierer mit Funktionen, die BASIC-Programmierer als selbstverständlich voraussetzen – der Möglichkeit, Speicherstellen untersuchen und verändern zu können.

Im nächsten Artikel führen wir das Projekt durch die ersten in diesem Artikel beschriebenen Entwicklungsstadien.

Steuerstrukturen

Die IF . . . THEN . . . ELSE-Struktur

Pseudo-Code	Assemblersprache
	DREI FCB 3
	.....
IF NUM1 = 3	IF LDA NUM 1
THEN	CMPA DREI
Routine 1	BNE ELSE
	THEN .....
	* Routine 1
ELSE	BRA ENDIF
Routine 2	ELSE .....
ENDIF	* Routine 2
	ENDIF .....

Die WHILE. . . WEND-Struktur

Pseudo-Code	Assemblersprache
WHILE NUM1 < = 3	WHILE LDA NUM1
wiederhole Routine	CMPA DREI
WEND	BGT WEND
	* wiederhole Routine
	BRA WHILE
	WEND .....

Die REPEAT . . . UNTIL-Struktur

Pseudo-Code	Assemblersprache
REPEAT	REPEAT .....
WIEDERHOLE Routine	* wiederhole Routine
UNTIL NUM1 < = 3	LDA NUM1
	CMPA DREI
	BGE REPEAT
	UNTIL .....

Die FOR . . . NEXT-Struktur

Pseudo-Code	Assemblersprache
FOR NUM1 = 1 TO NUM2	LDA NUM2
wiederhole Routine	FOR .....
NEXT NUM1	* wiederhole Routine
	DECA
	BGT FOR
	NEXT .....

Auf Fehlersuche

Als Programmbeispiel für die Technik „vom Ganzen zum Teil" konstruieren wir in Assembler einen „Debugger". Dabei entwickeln wir zunächst das Hauptmodul, das die einzelnen Aufgaben der Routinen in den niedrigeren Ebenen steuert.

Zunächst ein kurzer Blick auf die Aufgaben des Debuggers und seine Entwicklungsstufen. Das Programm soll folgende Eingaben empfangen können:

1) **Das zu bearbeitende Programm:** Wir gehen davon aus, daß das Programm sich beim Laden des Debuggers schon im Speicher befindet.

2) **Befehle:** Wir müssen entscheiden, welche Befehle direkt eingegeben und welche über ein Menü angeboten werden sollen. Wir nehmen dafür die Ein-Zeichen-Befehle des nebenstehenden Diagramms.

3) **Adressen:** Da die Eingabe in Hex erfolgt, muß der Hexadezimalstring im ASCII-Format in eine 16-Bit-Binärzahl umgewandelt werden.

Folgende Ausgaben werden erzeugt:

1) **Die „Echos" der eingegebenen Zeichen:** Bedenken Sie, daß das Drücken einer Taste nicht automatisch Zeichen auf den Schirm bringt – dafür ist ein Programm nötig.

2) **Zahlen im Acht- und 16-Bit-Format,** die als Hexadezimalstrings ausgegeben werden.

3) **Strings,** die als Labels für die unter Punkt 2 aufgeführten Zahlen dienen.

Ein Programm kann auf verschiedene Arten in Module und dann in Subroutinen zerlegt werden. Es muß jedoch ein Rahmenmodul vorhanden sein, das alle anderen Routinen aufruft.

B	Unterbrechungspunkt einsetzen
U	Unterbrechungspunkt löschen
D	Aktuelle Unterbrechungspunkte anzeigen
S	Programm starten
G	Nach einer Unterbrechung Programm an derselben Stelle fortsetzen
R	Registerinhalt anzeigen
M	Speicherstelle untersuchen und ändern
Q	Programm verlassen

Daten:
  Anfangsadresse des Programms (16 Bit)
  Prompt – für Befehlseingaben (das ASCII-Zeichen '>')
  Befehlszeichen – ein einzelnes ASCII-Zeichen (sollen hier auch Kleinbuchstaben möglich sein?)
  Unterbrechungsadresse – die Adresse der Routine, die den SWI-Interrupt handhabt.
Ablauf:
  Interrupt anlegen
  Anfangsandresse holen
  REPEAT
    Prompt anzeigen
    REPEAT
      Befehl holen
    UNTIL Befehl gültig ist
    Befehl darstellen (Echo)
    IF Befehl = 'B' THEN
      Unterbrechungspunkt einfügen
    ELSE IF Befehl = 'U' THEN
      Unterbrechungspunkt löschen
    ELSE IF . . .
  UNTIL Befehl = 'Q'
Ende des Hauptmoduls

Mehrere Subroutinen, die eine logische Einheit bilden, werden als Modul bezeichnet. So kann beispielsweise ein Modul das Einsetzen der Unterbrechungspunkte ausführen. Das nächste Stadium zeigt den Aufbau eines Moduls:

Daten:
Unterbrechungstabelle – ein Array von 16-Bit-Adressen, an denen die Unterbrechungsadressen gespeichert werden.
Entfernte-Werte – ein Array von Acht-Bit-Werten, die mit der obigen Tabelle zusammenhängt. Hier werden die durch SWI-Befehle ersetzten Unterbrechungspunkte gespeichert.
Zahl-der-Unterbrechungspunkte – ein Acht-Bit-Wert, in dem die Zahl der aktiven Unterbrechungspunkte gespeichert ist.
Nächster-Unterbrechungspunkt – ein Acht-Bit-Wert, der den nächsten Unterbrechungspunkt des Programms angibt.
SWI-Op-Code – ein Acht-Bit-Op-Code für den SWI-Befehl
Ablauf1: Unterbrechungspunkt-einfügen
IF Zahl-der-Unterbrechungspunkte < MAX
  THEN Nächster-Unterbrechungspkt. holen
  1 auf die Zahl-der-Unterbrechungspunkte addieren
  Adresse in der Unterbrechungstabelle speichern (Zahl-der-Unterbrechungspunkte)
ENDIF
Ende von Ablauf1
Ablauf 2: Unterbrechungspunkt(N)-anlegen
(N enthält die Information, welcher der Unterbrechungspunkte der Tabelle angelegt wird)
Aus der Unterbrechungstabelle(N)
Nächster-Unterbrechungspunkt holen
Den Op-Code dieser Adresse holen
In Entfernte-Werte(N) speichern
Für den Op-Code SWI-Adresse einsetzen
Ende von Ablauf 2

Ablauf 2 könnte jetzt schon codiert werden. Er hat folgende vier Datenwerte: Den Parameter N – eine Acht-Bit-Zahl von (1 bis Anzahl-der-Unterbrechungspunkte-1), die angibt, welcher Unterbrechungspunkt eingesetzt werden soll und als Offset für beide Tabellen dient. Beachten Sie, daß die eine Tabelle 16-Bit-Werte enthält und die andere Acht-Bit-Werte. Wir gehen

davon aus, daß N in A übergeben wird und die Adresse des Unterbrechungspunktes in X. Der ersetzte Op-Code wird zunächst in B zwischengespeichert und dann in die Tabelle Entfernte-Werte eingetragen.

Im nebenstehenden Kasten haben wir den Code für Ablauf2 aufgeführt (Modul „Unterbrechungspunkt-anlegen"). Wir müssen nun ein Modul für die Ein- und Ausgaben entwickeln. Wir gehen erst einmal davon aus, daß dafür zwei Subroutinen zur Verfügung stehen: INCH setzt ein einzelnes Zeichen von der Tastatur in das A-Register und OUTCH sendet das in A gespeicherte Zeichen an die aktuelle Cursorposition. Das Modul benötigt folgende Subroutinen:

1) BefehlHolen: Den nächsten Befehl von der Tastatur einlesen.
2) AdresseHolen: Eine Hex-Adresse von der Tastatur einlesen (ein bis vier Zeichen).
3) WertHolen: Einen Hex-Wert holen, der den Wert der Speicherstelle verändert (ein oder zwei Zeichen).
4) WertAnzeigen: Einen Hex-Wert aus zwei Zeichen auf dem Bildschirm anzeigen.
5) AdresseAnzeigen: Eine Hex-Adresse aus vier Zeichen auf dem Bildschirm anzeigen.

Hilfreiche Routinen

Bei unserem Verfahren sind die Unterschiede zwischen den Methoden „vom Ganzen zum Teil" und „vom Teil zum Ganzen" zu erkennen. Bei „Vom Ganzen zum Teil" definieren und codieren wir Vorgänge unabhängig voneinander und erhalten mehrere Routinen, die ähnliche Aufgaben ausführen. Die Methode „vom Teil zum Ganzen" spart jedoch Zeit, Aufwand und Platz, da wir nun einige wenige Routinen schreiben, die sich für mehrere Aufgaben einsetzen lassen. Hier die Definition dieser Routinen:

GETCH: Bei Eingabe eines einzelnen Zeichens in A das Zeichen mit der Liste der gültigen Zeichen vergleichen, gültige Zeichen (Befehl- oder Hexzeichen) darstellen und den Rest ignorieren.
GETHX2: Mit GETCH zwei Hex-Ziffern holen und in eine Acht-Bit-Zahl umwandeln.
GETHX4: Vier Hex-Ziffern für eine 16-Bit-Zahl holen.
PUTHEX: Eine Acht-Bit-Zahl als zwei Hex-Ziffern darstellen. (Kann für die Darstellung einer 16-Bit-Zahl zweimal aufgerufen werden.)
PUTCR: Ein Return (CR) ausgeben (oder – falls nötig – ein Return mit Zeilenvorschub).
Wir werden diese fünf Routinen nacheinander entwickeln. Hier der Aufbau von GETCH:

Daten:
Zeichen-ein ist ein per Tastatur eingegebenes ASCII-Zeichen (in A gespeichert)
Zeichen-OK enthält die 16-Bit-Adresse der Tabelle für gültige Zeichen
Zahl-der-Zeichen-OK ist ein Acht-Bit-Wert
Zeichen-gesucht ist ein Acht-Bit-Zähler

Ablauf:

```
REPEAT
   Nächstes Zeichen-ein holen
   Zeichen-gesucht auf (Zahl-der-Zeichen-
      OK –1) setzen
   WHILE Zeichen-OK(Zeichen-gesucht)
      <> Zeichen-ein
      AND Zeichen-gesucht >= 0
      Zeichen-gesucht dekrementieren
UNTIL Zeichen-gesucht >= 0
Zeichen-ein anzeigen
```

Für die Codierung dieses Ablaufs speichern wir Zeichen-Ein in A und die 16-Bit-Zahl Zeichen-OK in X. Zahl-der-Zeichen-OK kann zunächst an B übergeben werden, muß aber auf den Stack geschoben werden, damit sie nicht gelöscht wird. B läßt sich dann für Zeichen-gesucht einsetzen. Beachten Sie, daß B den Offset an die Tabelle übergibt und sich damit auch für die Identifizierung der Befehle und die Hexadezimalumwandlungen eignet.

Die GETCH-Routine

```
GETCH   PSHS    B       B speichern
REPTOO  BSR     INCH    Nächstes Zeichen-ein holen
        LDB     1,S     Zeichen-gesucht setzen
        DECB            Eins vom max. Offset subtrahieren
WHILOO  BLT     ENDWOO  WHILE Zeichen-gesucht >=0
        CMPA    B,X     AND
        BEQ     ENDWOO  Zeichen-ein <> Zeichen-OK
                        (Zeichen-gesucht)
        DECB            Zeichen-gesucht dekrementieren
        BRA     WHILOO
ENDWOO  TSTB
UNTLOO  BLT     REPTOO  UNTIL Zeichen-gesucht >=0
        BSR     OUTCH   Zeichen-ein anzeigen
        LEAS    1,S     S inkrementieren, um den
        RTS             ursprünglichen Wert von B zu
                        löschen
```

Das Modul „Unterbrechungspunkt-anlegen"

```
                        Definition der Daten
BPTAB   RMB     32      Tabelle der Unterbrechungspunkte
REMTAB  RMB     16      Entfernte Werte
NUMBP   FCB     0       Zahl-der-Unterbrechungspunkte
NEXTBP  FCB     0       Nächster-Unterbrechungspunkt
SWIOP   FCB     $3F     SWI-Op-Code
MAXBP   FCB     16      Max. Unterbrechungspunkte
                        Ablauf2 – Unterbrechungspunkt anlegen
BP02    PSHS    B,X     Zu ändernde Register sichern,
        LSLA            Den Offset mit 2 multiplizieren
        LEAX    BPTAB,PCR  Basisadresse der Tabelle
        LDX     A,X     Adresse (N) holen
        LDB     ,X      Op-Code dieser Adresse holen
        LSRA            A auf Urwert zurückstellen
        STB     A,X     Op-Code speichern
        LDB     SWIOP,PCR  SWI-Op-Code holen
        STB     ,X      In dieser Adresse speichern
        PULS    B,X,PC  Werte wiederherstellen/Return
                        Ende von Ablauf2
```

Unterbrechungen

Bei der Entwicklung unseres Debuggers vervollständigen wir das Modul zur Bearbeitung der Unterbrechungspunkte.

In unserem Unterbrechungspunktmodul fehlen noch die Subroutinen für das Löschen von Unterbrechungspunkten und das Wiedereinsetzen von Op-Codes, die zeitweilig durch SWI-Op-Codes ersetzt wurden. Als erstes entwickeln wir die Routine „Unterbrechungspunktlöschen" (aus der bereits bekannten Unterbrechungstabelle).

Die Unterbrechungstabelle (BPTAB) kann bis zu 16 Unterbrechungspunkte enthalten. Um einen bestimmten Punkt löschen zu können, müssen wir daher eine Zahl (zwischen 0 und 15) haben, die den Offset in der Tabelle angibt. Der Löschvorgang setzt alle auf diesen Punkt folgenden Einträge um eine Position (zwei Bytes) zurück und dekrementiert die Zahl-der-Unterbrechungspunkte.

Daten:

Zahl-der-Unterbrechungspunkte – ein Acht-Bit-Wert.

Zahl-Unterbrechungspunkt – ein Acht-Bit-Zähler

Unterbrechungstabelle – ein Array von 16-Bit-Adressen

Eintrag-zum-Löschen – ein Acht-Bit-Offset (mit einem Wert zwischen 1 und 16)

Ablauf:

```
Zahl-der-Unterbrechungspunkte dekrementieren
IF Eintrag-zum-Löschen (= Zahl-der-Unterbrechungspunkte(vorletzter) THEN
  FOR Zahl-Unterbrechungspunkt = Eintrag-zum-Löschen TO Zahl-der-Unterbrechungspunkte(vorletzter)
    Unterbrechungstabelle(Zahl-Unterbrechungspunkt + 1) auf Unterbrechungstabelle(Zahl-Unterbrechungspunkt) verschieben
    Entfernte-Werte(Zahl-Unterbrechungspunkt + 1) auf Entfernte-Werte (Zahl-Unterbrechungspunkt + 1) verschieben
  ENDFOR
ENDIF
```

Ende des Ablaufs

Der Parameter Eintrag-zum-Löschen kann an B übergeben werden. Dadurch erhält der Zähler, Zahl-Unterbrechungspunkt, der ebenfalls dort gespeichert wird, automatisch den richtigen Anfangswert. Er wird nach dem Vergleich mit Zahl-der-Unterbrechungspunkte automatisch dekrementiert, um den Offset in der Acht-Bit-Tabelle Entfernte-Werte bilden zu können. Danach wird er nach links verschoben (mit Zwei multipliziert), um den Offset in der 16-Bit-Unterbrechungstabelle anzugeben. Den Acht-Bit-Offset können wir in B speichern und den 16-Bit-Offset in A. Wenn die Adressen der Tabelleneinträge in X und Y gelegt werden, lassen sich die beiden Tabellen mit dem Auto-Inkrementbefehl durcharbeiten. Dabei können zwar die 16-Bit-Einträge über U versetzt werden, doch müssen wir für die Acht-Bit-Werte wieder A verwenden.

Die letzte Prozedur dieses Moduls nimmt den Unterbrechungspunkt aus dem Programm heraus, indem sie den SWI-Op-Code gegen den ursprünglichen, in der Tabelle Entfernte-Werte gespeicherten Op-Code austauscht.

Daten:

Zahl-Unterbrechungspunkt – ein Acht-Bit-Offset in die Unterbrechungstabelle

Ablauf:

```
Den Inhalt von Entfernte-Werte(Zahl-Unterbrechungspunkt) holen
In der Adresse der Unterbrechungstabelle (Zahl-Unterbrechungspunkt) speichern
```

Wir setzen voraus, daß der Parameter Zahl-Unterbrechungspunkt als Zahl zwischen 1 und 16 an B übergeben wurde und zum Tabellenoffset umgewandelt werden kann.

Wir beginnen nun mit dem Aufbau des Moduls, das die acht Steuerbefehle des Systems (je ein Zeichen) ausführt. Einige Befehle könnten die bereits codierten Routinen zwar direkt einsetzen, aus Gründen der Einheitlichkeit rufen wir sie im Steuermodul jedoch mit dem Befehl CALL auf.

Der Befehl B – das Einsetzen eines Unterbrechungspunktes – wird vollständig von der Routine Unterbrechungspunkt-Einsetzen (BP01) erledigt. Das Modul muß daher nur den folgenden Aufruf

```
CMDB BRA BP01
```

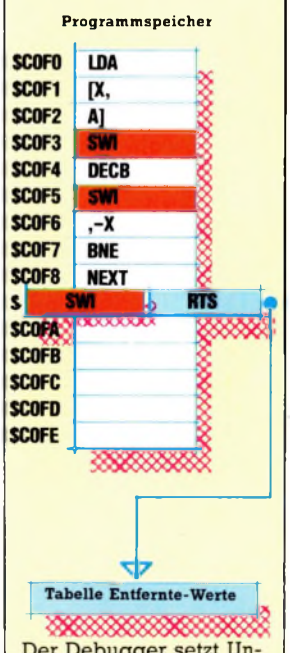
erhalten. Auch der Befehl U wird fast vollständig von der soeben entwickelten Routine (BP04) erledigt. Wir müssen dafür jedoch zuerst die Adresse des zu löschenden Unterbrechungspunktes erhalten und dann die Unterbrechungstabelle nach dieser Adresse durchsuchen. Ist die Adresse dort nicht vorhanden, wird der Befehl ignoriert; andernfalls übergeben wir den Offset ganz einfach an die Subroutine bei BP02.

Daten:

Prompt – wird dargestellt

Adresse-des-Unterbrechungspunktes – die Eingabe

Code(unter)brecher



Der Debugger setzt Unterbrechungspunkte in den analysierten Objectcode ein. Er speichert dafür zuerst den Code des Unterbrechungspunktes in der Tabelle Entfernte-Werte und inkrementiert den Zähler Zahl-der-Unterbrechungspunkte. Danach überschreibt er den Inhalt des Unterbrechungspunktes mit dem SWI-Op-Code. Beim Löschen wird der entfernte und in der Tabelle gespeicherte Wert wieder in den Programmcode eingesetzt und das überflüssige Tabellenbyte gelöscht, indem alle darauffolgenden Einträge um eine Position rückwärts verschoben werden.

Unterbrechungstabelle
Zahl-Unterbrechungspunkt
Ablauf:
Prompt darstellen
Adresse-des-Unterbrechungspunktes holen
Zahl-Unterbrechungspunkt auf 16 setzen
WHILE Unterbrechungstabelle(Zahl-Unterbrechungspunkt) <> Adresse-des-Unterbrechungspunktes AND Zahl-Unterbrechungspunkt > 0
Zahl-Unterbrechungspunkt dekrementieren
IF gefunden THEN
Unterbrechungspunkt herausnehmen

Die Adresse-des-Unterbrechungspunktes bleibt in Y, damit X als Zeiger auf die Tabelle eingesetzt werden kann. Zahl-Unterbrechungspunkt bleibt in B.

Der Befehl D zeigt die Unterbrechungspunkte. Er wird von der Routine DISPBP erledigt und durch den Aufruf einer Subroutine angesprochen:

CMDD BRA DISPBP

Der Befehl S löst den Start des Programms aus und aktiviert die Unterbrechungspunkte. Dabei wird der SWI-Op-Code in die Adressen der Unterbrechungstabelle eingesetzt, der ursprüngliche Op-Code dieser Position in Entfernte-Werte gespeichert und die Steuerung an die Anfangsadresse des Programms übergeben. Beachten Sie, daß der nächste Unterbrechungspunkt die Zahl 1 ist. Hier der Ablauf für den Programmstart:

Daten:
Zahl-der-Unterbrechungspunkte – ein Acht-Bit-Wert
Unterbrechungstabelle
Entfernte-Werte
Zahl-Unterbrechungspunkt – ein Acht-Bit-Zähler
Nächster-Unterbrechungspunkt – ein Acht-Bit-Wert
SWI-Op-Code – ein Acht-Bit-Wert
Anfangsadresse – die 16-Bit-Adresse des analysierten Programms
Ablauf:
Zahl-Unterbrechungspunkt auf Zahl-der-Unterbrechungspunkte setzen
WHILE Zahl-Unterbrechungspunkt > 0
Unterbrechungspunkt-Einrichten(Zahl-Unterbrechungspunkt)
ENDWHILE
Nächster-Unterbrechungspunkt auf 1 setzen
Auf Anfangsadresse springen

Für diese Aufgabe können wir die bereits codierte Routine Unterbrechungspunkt-einsetzen verwenden, die den Wert Zahl-Unterbrechungspunkt (minus Eins, damit er als Tabellenoffset dienen kann) in A erwartet. Wir werden daher A dekrementieren, bevor wir Unterbrechungspunkt-einsetzen aufrufen. Auf der nächsten Seite ist der Code dieser Routine abgedruckt.

Das Ende der Routine ist erklärungsbedürftig. Da beim Ablauf des getesteten Programms keine zusätzlichen Werte auf dem Stack liegen dürfen, müssen wir sicherstellen, daß der Stack bei Übergabe der Steuerung an das Programm leer ist. Nun kann das Hauptmodul zwar alle überflüssigen Werte leicht vom Stack herunterziehen, doch läßt sich nicht verhindern, daß bei einem BSR-Aufruf (den wir aus Gründen der Einheitlichkeit einsetzen) die Rücksprungadresse auf den Stack geschoben wird. Wenn wir diesen Wert dort belassen und das Programm oft neu starten, wächst der Stack ständig. Zur Lösung dieses Problems löschen wir diese Adresse in dem Moment, in dem wir auch die Steuerung an das Programm zurückgeben. Wir ersetzen dabei einfach die Rücksprungadresse durch die Anfangsadresse des Programms. RTS zieht dann als Rücksprungadresse die Startadresse vom Stack: Die Steuerung ist übergeben, während gleichzeitig der Stack bereinigt wird.

Mit dem Befehl M lassen sich Speicherstellen untersuchen und verändern. Dafür muß eine Eingabeadresse geholt und der Inhalt dieser Adresse auf dem Bildschirm angezeigt werden. Der Anwender kann entweder eine neue zweistellige Hexzahl auf dieser Adresse speichern oder ein Return oder einen Punkt eingeben. In den beiden ersten Fällen müssen wir die darauffolgende Speicherstelle ansprechen, während ein Punkt den Ablauf abbricht.

Daten:
Aktuelle-Adresse – die 16-Bit-Adresse der untersuchten Speicherstelle
Aktueller-Wert – der Inhalt von Aktuelle-Adresse (ein Acht-Bit-Wert)
Neuer-Wert – ersetzt Aktueller-Wert. Er hat ebenfalls ein Acht-Bit-Format
Ablauf:
Aktuelle-Adresse holen
REPEAT
Aktueller-Wert anzeigen
Neuer-Wert holen
IF Neuer-Wert kein Punkt THEN
IF Neuer-Wert kein Return THEN
Neuer-Wert in Aktuelle-Adresse speichern
ENDIF
Aktuelle-Adresse inkrementieren
Aktuelle-Adresse anzeigen
ENDIF
UNTIL Aktuelle-Adresse ein Punkt ist

Für diese Befehlsroutine wird Aktuelle-Adresse in X abgelegt und das B-Register sowohl für Aktueller-Wert und Neuer-Wert eingesetzt. A enthält das Flag, das anzeigt, welche der drei Alternativen (Hexzahl, Punkt oder Return) eingegeben wurde.

In der nächsten Folge codieren wir die letzten drei Befehle G, R und Q und untersuchen den dabei eingesetzten Interrupt-Mechanismus. Außerdem entwickeln wir das Hauptmodul unseres Debuggers.

Die Routine „Unterbrechungspunkt-löschen“

```
BP04    PSHS    A,B,X,Y,U       Eingesetzte Register sichern
        DEC     NUMBP,PCR       Zahl der-Unterbrechungspunkte
                                dekrementieren

IF02    CMPB    NUMBP,PCR       IF Eintrag-zum-Löschen <
        BGT     ENDF02            Zahl-der-Unterbrechungspunkte
        DECB                    B in Offset umwandeln
        TFR     B,A             B nach A kopieren
        LSLA                    A in Offset umwandeln
        LDX     BPTAB,PCR       Basisadresse der Unterbrechungs-
                                tabelle
        LEAX    A,X             Unterbrechungstabelle(Zahl-
                                Unterbrechungspunkt)
        LDY     REMTAB,PCR      Basisadresse von Entfernte-Werte
        LEAY    B,Y             Entfernte-Werte(Zahl-
                                Unterbrechungspunkt)
FOR00   LDU     2,X             Eintrag der Unterbrechungs-
                                tabelle holen, der verschoben wird
        STU     ,X++            Um eine Position zurücksetzen
        LDA     1,Y             Den Eintrag von Entfernte-Werte
                                holen, der verschoben werden soll
        STA     ,Y+             Um eine Position zurücksetzen
        INCB
        CMPB    NUMBP,PCR       War das der letzte?
        BLT     FOR00           Nächster
ENDF02  PULS    A,B,X,Y,U,PC    Register wiederherstellen und
                                Rücksprung
```

Die Routine „Unterbrechungspunkt-herausnehmen“

```
BP05    PSHS    A,B,X
        DECB                    In den Offset für Entfernte-Werte
                                umwandeln
        LDX     REMTAB,PCR      Basisadresse von Entfernte-
                                Werte
        LDA     B,X             Wert holen, der verschoben wird
        LSLB                    In den Offset für die Unterbre-
                                chungstabelle umwandeln
        LDX     BPTAB,PCR       Basisadresse der Unterbre-
                                chungstabelle
        STA     [B,X]           Auf Tabellenadresse speichern
        PULS    A,B,X,PC        Register wiederherstellen und
                                Rücksprung
```

Der Befehl U

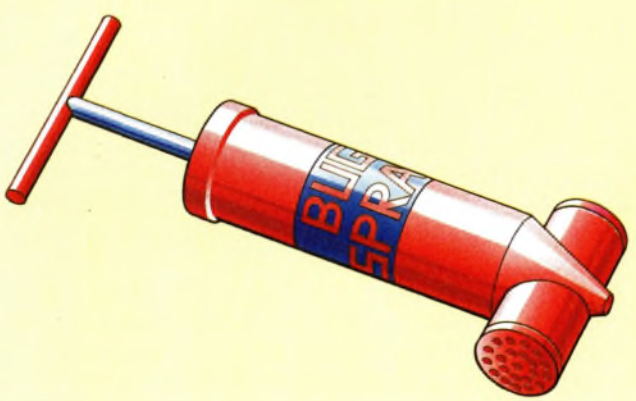
```
PROMPT  FCB     '>
CMDU    PSHS    A,B,X,Y         Eingesetzte Register sichern
        LDA     PROMPT,PCR
        BSR     OUTCH           Prompt anzeigen
        BSR     GETADD          Adresse holen
        TFR     D,Y             Adresse in Y speichern
        LDB     MAXBP,PCR       Größtmögliche Zahl von Unterbre-
                                chungspunkten (16)
        LDX     BPTAB,PCR       Basisadresse der Unterbrechungs-
                                tabelle
        TFR     B,A
        LSLA                    A ist Offset für das Ende der
                                Unterbrechungstabelle
        LEAX    A,X             X zeigt nun über das Tabellenende
                                hinaus
        TSTB                    Je nach Inhalt von B Flags setzen
WHIL02  BLE     ENDW02          WHILE B > 0
        CMPY    ,--X            (Bedenken Sie, daß X zuerst dekre-
                                mentiert wurde)
        BEQ     ENDW02          AND die Adresse-des-
                                Unterbrechungspunktes nicht
                                vorhanden ist
        DECB                    Zahl-Unterbrechungspunkt dekre-
                                mentieren
        BRA     WHIL02
ENDW02  TSTB                    Gefunden, wenn B > 0
IF03    BLE     ENDF            IF gefunden THEN
        BSR     BP04            Unterbrechungspunkt-heraus-
                                nehmen
ENDF03  PULS    A,B,X,Y
```

Der Befehl S

```
START   RMB     2               Startadresse
CMDS    LDA     NUMBP,PCR       Zahl-Unterbrechungspunkt auf
                                Zahl-der-Unterbrechungspunkte
WHIL03  TSTA                    Den Wert von Zahl-Unterbre-
                                chungspunkt testen
        BLE     ENDW03          WHILE Zahl-Unterbrpkt > 0
        DECA                    Zahl-Unterbrpkt dekrementieren
        BSR     BP02            Unterbrechungspunkt-anlegen
        BRA     WHIL03          Nächster Unterbrechungspunkt
ENDW03  LDA     #1              Nächster-Unterbrechungspunkt auf
        STA     NEXTBP,PCR      1 setzen
        STD     1,S
        RTS
```

Der Befehl M

```
PROMPT  FCB     '>
SPACE   FCB     32              ASCII-Code für Leerzeichen
CMDM    PSHS    A,B,X           Eingesetzte Register sichern
        LDA     PROMPT,PCR
        BSR     OUTCH           Prompt anzeigen
        BSR     GETADD          Aktuelle-Adresse holen
        TFR     D,X             Auf X verschieben
REPT01  LDB     ,X              Aktueller-Wert holen
        BSR     PUTHEX          Aktueller-Wert anzeigen
        LDA     SPACE,PCR
        BSR     OUTCH           Leerzeichen anzeigen
        BSR     GETVAL          Neuer-Wert holen
IF03    TSTA                    IF Neuer-Wert kein Punkt
        BLT     UNTL01
        BGT     ENDF03          IF Neuer-Wert kein Return
        STB     ,X              Neuer-Wert in Aktuelle-Adresse
                                speichern
ENDF03  LEAX    1,X             Aktuelle-Adresse inkrementieren
        TFR     X,D             Aktuelle-Adresse anzeigen
        BSR     DSPADD
        BRA     REPT
UNTL01  PULS    A,B,X,PC
```



Des Fehlers Tod

Wir beschließen unsere Serie über die Assemblersprache des 6809. Wir beenden alle unerledigten Aufgaben, geben einen Überblick über den Befehlsfluß und codieren das Hauptmodul.

Unser Hauptmodul aktiviert als erstes den Interruptmechanismus, mit dem wir Unterbrechungspunkte in das analysierte Programm einsetzen. Die Unterbrechungspunkte übergeben die Steuerung an den Debugger und ermöglichen es uns, die Inhalte der Register und Speicherstellen zu untersuchen. Über die Anfangsadresse des getesteten Programms geben wir die Steuerung wieder dorthin zurück (Befehl S). Die Befehle G und S übergeben die Steuerung an das getestete Programm, der Debugger dagegen erhält die Kommandos von den SWI-Codes der Unterbrechungspunkte.

Zwei Teile der Initialisierung hatten wir schon in der letzten Folge codiert. Dabei stand der Einsprungspunkt für Interrupts unmittelbar hinter dem Aufruf der Initialisierungsroutine. Unser erster Befehl sicherte den Stack-Pointer S, der später als Bezugspunkt für die von SWI auf den Stack geschobenen Registerwerte dient. Der nächste Initialisierungsschritt ermöglicht nun die Interpretation der Befehle.

Die Subroutinen für die Ausführung der Befehle sind fertig – ihr Aufruf (je nach Befehlseingabe) muß jedoch noch programmiert werden. Diese Aufgabe ließe sich durchaus mit einer Reihe verschachtelter IF-Befehle lösen, doch setzt die Routine „Befehl-Holen" einen Offset in die Tabelle der Befehlszeichen, den wir mit einer Sprungtabelle einsetzen können. Diese Methode ist an dieser Stelle zwar nicht optimal, doch lohnt es sich, diese praktische Technik zu zeigen. Wir müssen dazu eine Sprungtabelle mit den Adressen der Subroutinen anlegen.

JMP kann – im Gegensatz zu Verzweigungsbefehlen – alle normalen Adreßarten einsetzen, darunter auch die indizierte und die indirekte Adressierung. Wenn wir X mit der Basisadresse der Tabelle laden und den Offset in B verwenden (in verdoppeltem Format, da die Tabelle 16-Bit-Werte enthält und nicht die Acht-Bit-Werte der Befehle), überträgt der Befehl

JMP [B,X]

die Steuerung auf die entsprechende Subroutine. Der BSR-Aufruf nimmt dabei die Adresse dieses Sprungbefehls an. Wir müssen nun einen zweiten Initialisierungsschritt anlegen.

Daten:
- Sprungtabelle – eine Tabelle mit acht 16-Bit-Adressen
- CMDB, CMDU etc. die Startadressen der Subroutinen

Ablauf:
```
FOR jede Subroutine
   Startadresse holen
   Startadresse in der Sprungtabelle
      speichern
ENDFOR
```

Wir müssen nun überlegen, was bei Eingabe des Befehls Q (Programm beenden) geschehen soll. Das Programm und der Debugger sollten so zurückgelassen werden, daß sie für weitere Aufrufe ohne irgendwelche Schwierigkeiten zur Verfügung stehen.

Dafür muß der Stack am Programmende aber im gleichen Zustand sein wie am Anfang. Wir könnten nun für unser Programm einen separaten Stack anlegen, indem wir den Wert von S ändern und am Ende einfach den alten Wert wieder einsetzen. Diese Technik ist sehr praktisch – in unserem Fall könnten wir jedoch Schwierigkeiten haben, ungenutzten Speicherplatz zu finden, da der RAM-Bereich bereits mit zwei Programmen belegt ist. Es wäre auch möglich, S einfach entsprechend zu inkrementieren und so alle überflüssigen Werte zu löschen. Doch auch dieser Vorgang bietet Probleme, da wir nicht wissen, ob Interrupts aufgetreten sind und sich dadurch die Stackinhalte verändert haben. Am einfachsten wäre es, den Anfangswert von S zu speichern und mit dem letzten Befehl des Programms wieder einzusetzen.

Ein Zwischenspeicher

Aus diesem Grund speichert der in der Initialisierung eingerichtete Interruptmechanismus drei Byte an der Adresse, die der SWI-Vektor bei $FFFA angibt. Auch hier müssen wir den ursprünglichen Wert wieder einsetzen, da ein Interrupt des Betriebssystems unvorhergesehene Abläufe auslösen kann. Wir brauchen also einen weiteren Initialisierungsteil, der alle Werte speichert, die von der Endroutine wieder eingesetzt werden sollen.

Daten:
- Gesichert – fünf Byte mit den ursprünglichen Werten
- Stack-Pointer – der aktuelle Wert von S, plus 2
- SWI-Vektor – der Wert von $FFFA

Ablauf:
Stack-Pointer in Gesichert speichern

SWI-Vektor holen
Die drei Byte beim SWI-Vektor in Gesichert speichern

Die Endroutine (der Befehl Q) muß diesen Vorgang nur umkehren und die Steuerung an das Betriebssystem zurückgeben. Für die Ausführung dieser Vorgänge gibt es wieder eine Reihe von Möglichkeiten: Nach der Wiederherstellung des Vektors kann entweder der SWI-Befehl verwandt oder aber ein Sprung auf einen bekannten Punkt des Betriebssystems ausgeführt werden.

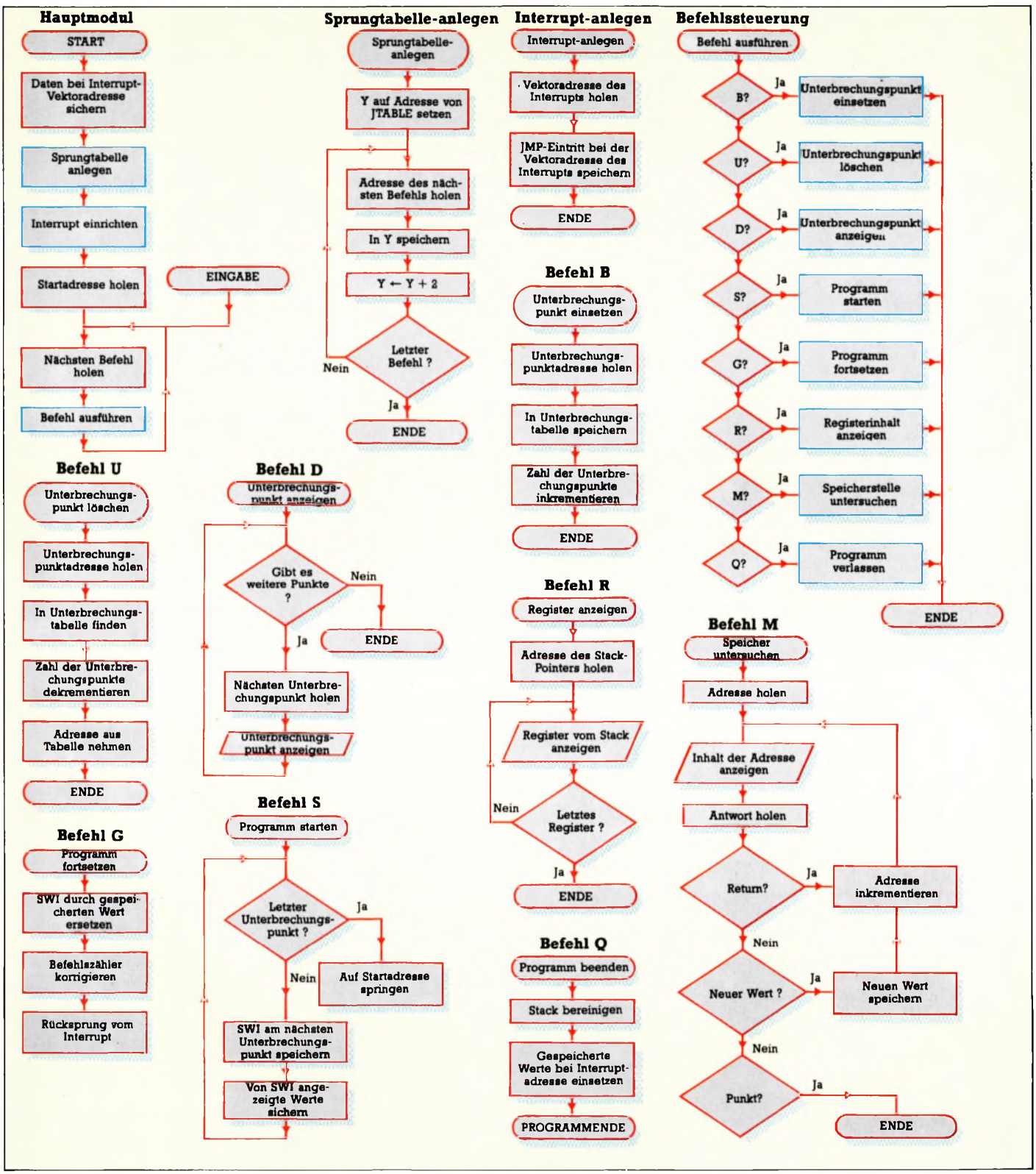
Daten:

Gesichert – fünf Byte mit den ursprünglichen Werten

Stack-Pointer – der aktuelle Wert von S, plus 2

SWI-Vektor – der Wert bei $FFFA

Die Ablaufdiagramme zeigen den Programmfluß der Debuggermodule. Sie sind in der Reihenfolge aufgeführt, in der sie von anderen Routinen aufgerufen werden. Blau umrahmte Kästen zeigen an, daß separate Routinen aufgerufen werden.



Reset-Vektor – der Wert bei $FFFE

Ablauf:

Die drei Byte des SWI-Vektors mit Gesichert wiederherstellen
Stack-Pointer wiederherstellen
Zum Betriebssystem verzweigen

Wir wenden uns nun dem Hauptmodul zu.

Daten:

Prompt für die Befehlseingabe ist das ASCII-Zeichen '>'
Befehls-Offset zeigt in die Befehlstabelle und die Sprungtabelle

Ablauf:

Werte-sichern
Sprungtabelle-anlegen
Interrupt-einrichten
Start-Adresse holen
REPEAT
Prompt darstellen
Befehl-holen
Befehl-ausführen
Kein Ende

Damit ist unser Debuggerprogramm fertig. Sie können den Code jedoch noch optimieren und abkürzen. So müssen beispielsweise viele Werte bewegt werden, damit sie für die Subroutinen in den korrekten Registern stehen. Eine Umstellung der Register könnte die Abläufe vereinfachen, wäre aber nur zu empfehlen, wenn der Speicherplatz sehr begrenzt ist. Weiterhin haben wir die gleichen Daten an mehreren Stellen definiert. Es gibt jedoch zwei Methoden, Daten in einem Programm einzusetzen: Die Daten können entweder innerhalb der eingesetzten Module bleiben – das ist theoretisch der beste Weg – oder vollständig am Anfang des Programms definiert werden. Dies ist besonders dann vorteilhaft, wenn Sie einen Disassembler mit dem Programm einsetzen wollen.

Der Debugger sollte in einen Speicherbereich geladen werden, der nicht von dem zu analysierenden Programm belegt ist. Da das Programm per Sprung auf die DEBUG-Einsprungadresse aufgerufen wird, muß diese Adresse vor dem Programmstart bekannt sein.

Durch diese Serie sollten Sie umfassendes Verständnis über Assemblerprogrammierung und Entwicklungsmethoden bekommen haben.

Sprungtabelle-anlegen

```
JTABLE  RMB   16          Platz für acht Zwei-Byte-Adressen

SETUPJ  LEAY  JTABLE,PCR  Basisadresse der Tabelle in Y
        LEAX  CMDB,PCR    Startadresse der CMDB-Subroutine
        STX   ,Y++        In der Tabelle speichern
        LEAX  CMDU,PCR    Startadresse der CMDU-Subroutine
        STX   ,Y++        In der Tabelle speichern
        LEAX  CMDD,PCR    Startadresse der CMDD-Subroutine
        STX   ,Y++        In der Tabelle speichern
        LEAX  CMDS,PCR    Startadresse der CMDS-Subroutine
        STX   ,Y++        In der Tabelle speichern
        LEAX  CMDG,PCR    Startadresse der CMDG-Subroutine
        STX   ,Y++        In der Tabelle speichern
        LEAX  CMDR,PCR    Startadresse der CMDR-Subroutine
        STX   ,Y++        In der Tabelle speichern
        LEAX  CMDM,PCR    Startadresse der CMDM-Subroutine
        STX   ,Y++        In der Tabelle speichern
        LEAX  CMDQ,PCR    Startadresse der CMDQ-Subroutine
        STX   ,Y++        In der Tabelle speichern
```

Hier findet der eigentliche Sprung zu den Subroutinen statt. Wir gehen davon aus, daß X die Adresse von JTABLE enthält und B das Offset

```
DOCMD   JMP   [B,X]
```

Werte-Sichern

```
SAVED   RMB   5           Die fünf zu sichernden Byte
SAVEIT  LEAX  SAVED,PCR   Speicheradresse
        TFR   S,D         S nach D verlegen
        ADDD  #2          Zwei addieren (für Rücksprungadresse)
        STD   ,X++        Sichern
        LDY   $FFFA       Interrupt-Vektoradresse holen
        LDA   ,Y+         Erstes zu sichernde Byte holen
        STA   ,X+         Sichern
        LDD   ,Y          Die anderen beiden Byte holen
        STD   ,X          Sichern
        RTS
```

Befehl Q

```
CMDQ    LEAX  SAVED,PCR   Adresse von Gesichert
        LDY   $FFFA       SWI-Vektor
        LDA   2,X         Erstes von drei Byte
        STA   ,Y+         Wiederherstellen
        LDD   3,X         Die anderen beiden Byte
        STD   ,Y          Wiederherstellen
        LDS   ,X          Stack-Pointer sichern
        JMP   [$FFFE]     Indirekter Sprung über den Reset-Vektor
```

Hauptmodul

```
PROMPT  FCB   '>
STACKP  RMB   2           Stack-Pointer für Register-anzeigen

DEBUG   BSR   SAVEIT      Werte-sichern
        BSR   SETUPJ      Sprungtabelle-anlegen
        BSR   INIT        Interrupt-einrichten und Start-adresse-holen

ENTRY   STS   STACKP,PCR  Stack-Pointer sichern
        LEAX  JTABLE,PCR
REPT02  LDA   PROMPT,PCR  Prompt holen und
        BSR   OUTCH       anzeigen
        BSR   GETCOM      Befehl holen
        LSLB              Doppelter Offset für 16-Bit-Tabelle
        BSR   DOCMD       Befehl ausführen
        BRA   REPT02      Nächster Befehl
```

System in Betrieb

Betriebssysteme sind Schnittstellen zwischen Hardware und Programmen. Mit diesem Artikel beginnt eine Serie, in der wir die Betriebssysteme bekannter Heimcomputer untersuchen.

Betriebssysteme (Operating System=OS) sind in einer Maschinensprache geschrieben, die dem Microprozessor des gesteuerten Computers entspricht. So wurde das Betriebssystem des Acorn B im Maschinencode des 6502 entwickelt und das des Spectrum selbstverständlich im Z80-Code.

Ein OS besteht aus Routinen, die vielfältige Maschinenfunktionen ausführen. Eine Routine überprüft beispielsweise ständig, ob Tasten gedrückt wurden. Abläufe dieser Art braucht der Anwender daher nicht erst zu entwickeln. Mit einem guten OS sollte es sogar möglich sein, jede Hardwarefunktion anzusprechen, ohne daß die exakten Speicherpositionen der einzelnen Routinen oder Speichertabellen bekannt sein müssen. Auch Hardwareänderungen lassen sich per OS leichter integrieren. Dabei wird das Betriebssystem so verändert, daß die Programme älterer Maschinen ebenfalls laufen.

Das Betriebssystem des Acorn B ist durchdacht aufgebaut. Ein Programm, das auf einem Standardgerät im BASIC des Acorn B geschrieben wurde, funktioniert auch auf einer Maschine, die mit einem zweiten Prozessor ausgerüstet ist – trotz der umfassenden Hardwareänderung. Das OS des Acorn B mußte allerdings mehrere Entwicklungsstadien durchlaufen, bevor es seine augenblickliche hochentwickelte Form erhielt. Die erste Version – 0.1 – bestand aus vier EPROM-Chips, sie ist inzwischen fast vollständig vom Markt verschwunden. 0.1 war nicht ausreichend flexibel und konnte keine Diskettenlaufwerke ansprechen. Die Version 1.0 unterstützte die Speicherung auf Diskette. Sie bestand aus einer kleinen Platine mit zwei Acht-KByte-Chips im Inneren des Geräts. Die aktuelle Version – 1.2 – ist heute auf fast allen Acorn-B-Geräten zu finden. Eine Variante davon wurde speziell für den amerikanischen Markt entwickelt.

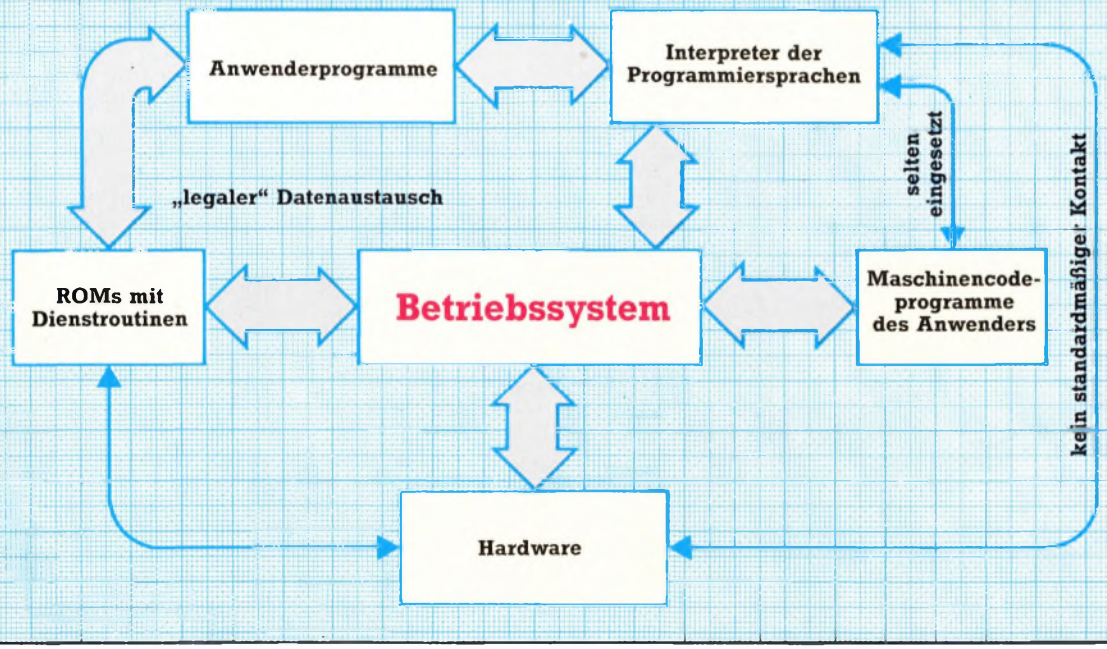
Blick ins Innenleben

Bei der Eingabe von *FX0 und Return erscheint die Systemnummer Ihrer Maschine auf dem Bildschirm. Der Befehl *HELP zeigt außer der OS-Nummer auch die Namen der eingebauten ROM-Chips.

Die Betriebssystemaufgaben des Acorn B lassen sich in vier Hauptbereiche unterteilen:
1) **Eingaberoutinen:** Diese Routinen nehmen Informationen aus dem „Current Input Stream" auf. Normalerweise wird dabei die Tastatur angesprochen, doch auch die RS423-Schnittstelle und das Dateisystem (das mit dem Befehl *EXEC aktiviert wird) gehören in diese Kate-

Betrieb läuft

Bei einem guten Betriebssystem merkt man nicht, daß es arbeitet. Für den Anwender sollte die Schnittstelle zwischen Programmen und Hardware (das heißt die Steuerung der Diskettenlaufwerke, das Zwischenspeichern von Ausgaben und die Bildschirmanzeige etc.) völlig einsichtig sein. Weiterhin sollte das OS der Soft- und Hardware ständig korrekt formatierte und zeitlich aufeinander abgestimmte Ein- und Ausgaben bieten.



gorie. Die wichtigste OS-Routine der Eingabeverarbeitung heißt OSRDCH (Operating System ReaD CHaracter – OS-Zeichen einlesen).
2) Routinen für Ausgabe und Anzeige: Diese Routinen steuern die Ausgaben des Computers. Der Acorn B kann eine ganze Reihe von Ausgabemöglichkeiten als „Current Output Stream" einsetzen, beispielsweise den Bildschirm, Drucker, die RS423-Schnittstelle und (über den Befehl *SPOOL) das aktuelle Dateisystem. Außer Druck und Datenausgabe steuern diese Routinen unter anderem auch den 6845-Bildschirmchip des Computers und den Einsatz der vom Anwender definierten Zeichen. Das OS ruft dafür OSWRCH (Operating System WRite CHaracter – OS-Zeichen darstellen), OSASCII und OSNEWL auf.
3) Dateisysteme: Jedes Betriebssystem muß dem Anwender die Möglichkeit geben, den Inhalt des Arbeitsspeichers auf ein permanentes Speichermedium zu sichern. Dieser Betriebssystembereich wird „Aktuelles Dateisystem" genannt, wobei der Acorn B die Wahl zwischen Cassette, Diskette, Econet, ROM oder Telesoftware bietet. Die OS-Routinen haben außerdem zusätzliche ROM-Module zur Verfügung, mit denen das OS die Hardware der einzelnen Dateisysteme steuern kann.

Um über die ROMs des seitenbezogenen Dateisystems einfache Schnittstellenverbindungen zu den magnetischen Speichermedien herstellen zu können, wurden einige Standardroutinen des OS für die Dateispeicherung abgestellt. Darunter gibt es Module, die ganze Dateien lesen oder schreiben, einzelne Bytes aus einer offenen Datei lesen oder Bytegruppen auf dem Speichermedium ansprechen. Die sieben Submodulaufrufe des Betriebssystems, die für Dateien zuständig sind, zeigen mit Vektoren auf die entsprechenden Subroutinen des Cassettensystems. Ein ROM mit seitenbezogenem Dateisystem kann diese Vektoren verändern und so die eigenen Dateiprogramme für BASIC oder Assembler einsetzen. Mit einem ROM dieses Typs (DFS ROM) kann der Acorn B Diskettenlaufwerke ansprechen.
4) Interrupts: Interrupts sind von der Hard- oder Software erzeugte Signale, die die CPU veranlassen, ihre aktuelle Aufgabe zu unterbrechen und Vorgänge auszuführen, die im Augenblick wichtiger sind. Danach nimmt die CPU ihre ursprüngliche Arbeit wieder auf. Der Acorn B besitzt eine Reihe Interrupt-gesteuerter Routinen, die dem Anwender über OS zur Verfügung stehen.

Zwei heiße Drähte

Außer diesen vier Hauptbereichen gibt es noch die beiden Systemaufrufe OSBYTE und OSWORD, die eine große Anzahl unterschiedlicher Funktionen steuern: den Tonerzeugungschip, die Break-Taste etc.

In den meisten Fällen steht der Aufbau des Arbeitsspeichers und der Hardware schon wenige Monate nach Vorstellung einer neuen Maschine zur Verfügung. Doch wenn diese Einzelheiten bekannt sind, warum sollen wir uns dann überhaupt um Betriebssystemaufrufe kümmern, und warum können wir Funkionseinheiten oder Speicher nicht ganz einfach und viel besser direkt ansprechen?

Große Flexibilität

Einen Teil der Antwort haben wir schon am Anfang des Artikels gegeben: Mit OS-Aufrufen sind die Programme gegenüber Änderungen flexibler, die der Hersteller in Hardware und Konfiguration vornimmt. Umgekehrt verursacht der Einsatz fester ROM-Adressen große Probleme, wenn das ROM umgestellt wird. Beim Aufruf von ROM-Routinen über das OS brauchen Sie Änderungen nicht zu beachten.

Der folgende Befehl schreibt auf einem Acorn B den Wert 200 an den User Port bei der Adresse &FE60.

```
?&FE60=200
```

Verfügt das Gerät jedoch über einen zweiten Prozessor, dann wird der Wert nicht an den User Port, sondern in eine Speicherstelle des zweiten Prozessors gesandt. Mit dem entsprechenden OS-Aufruf läßt sich das Problem jedoch umgehen. Die Routine „weiß" dann, wie der Wert zum User Port gesandt wird, unabhängig davon, ob ein zweiter Prozessor vorhanden ist oder nicht:

```
*FX 151,96,200
```

Es gibt mehrere Aufrufe, mit denen man sich auf dem Acorn B beispielsweise Zugang zu dem VIA (Versatile Interface Adaptor), dem System-VIA oder aber auch dem 1Megahertz-Bus verschaffen kann.

Warum also das Rad nochmals erfinden? Wenn eine Routine für eine bestimmte Maschinenfunktion bereits existiert, brauchen die ROM-Routinen nicht direkt angesprochen zu werden. Betriebssystemaufrufe sind im allgemeinen leistungsfähiger als der direkte Zugriff auf ROM-Routinen.

Nur wenn man hohe Verarbeitungsgeschwindigkeiten braucht oder für eine bestimmte Aufgabe keine OS-Routinen existieren, sollten Sie direkt mit dem Speicher oder der Hardware arbeiten, da sich hohe Geschwindigkeiten oft nur durch direkten Zugriff und nicht über OS-Routinen erreichen lassen. Dieser Weg ist jedoch nicht einfach, und Programme, die auf einer bestimmten Maschine laufen, funktionieren unter Umständen nicht auf Geräten mit nur geringfügig geänderter Hardware oder einer anderen OS-Version.

In der nächsten Folge sehen wir uns die OS-Routinen des Acorn B im einzelnen an. Den Spectrum und andere Maschinen untersuchen wir zu einem späteren Zeitpunkt.

Die Memory Map des Acorn B

In dieser Folge untersuchen wir, wie der Acorn B seinen Speicher einsetzt und wie ein Betriebssystem mit Vektoren arbeitet.

Viele Speicherbereiche des Acorn B führen mehrere Funktionen aus. Ein typisches Beispiel ist die Speicherseite &9 (ein RAM-Bereich zwischen &900 und &9FF), die zu unterschiedlichen Zeiten als RS232-Ausgabebuffer, Cassettenausgabebuffer oder Arbeitsbereich für Sprache oder Tongenerierung dient.

Der Block von &0E00 bis &1900 ist ein weiteres Beispiel. Auf einem Acorn B mit Cassettenrecorder als Dateisystem steht dieser Speicherbereich für BASIC-Programme zur Verfügung. Beim Anschluß eines Diskettensystems dient er jedoch als Arbeitsbereich und verringert dadurch den Platz für BASIC-Programme um über 2,5 KByte. In den Modi 0 und 2 ist das besonders lästig, da dort der Speicher bereits eingeschränkt ist.

Auch der Block zwischen &8000 und &BFFF wird mehrfach genutzt. Hier liegt der Einsatzbereich für das DFS-ROM, das Textsystem oder ein ROM mit Dienstroutinen, aber auch der vom ROM des BASIC-Interpreters angesprochene Speicherbereich. Die „Paged"-ROMs lassen sich nur einzeln aktivieren. Dabei wählt das Betriebssystem das ROM und schaltet es ein. Normalerweise ist das BASIC-ROM mit dem BASIC-Interpreter aktiv, so daß BASIC-Programme sofort eingegeben und ausgeführt werden. Bei einem DFS-Befehl schaltet das OS jedoch das DFS-ROM ein, führt den Befehl aus und aktiviert wieder das BASIC-ROM. Während der gesamten Ausführung des DFS-Befehls wird das BASIC-ROM ignoriert. Andere ROMs, beispielsweise das Telesoftware Dateisystem oder der Textchip Wordwise, belegen ebenfalls diesen Speicherbereich und schalten sich auf Anforderung ein oder aus. Ohne das „Paging" würde nur ein sehr kleiner Arbeitsspeicher zur Verfügung stehen.

Nachdem das Betriebssystem seinen Speicher belegt hat, steht uns der Rest zur Verfügung. Die BASIC-Variable PAGE enthält die Startadresse des Programmbereichs für BASIC-Texte, während HIMEM auf den Anfang des Bildschirmspeichers zeigt. Der Platz zwischen diesen beiden Punkten steht für BASIC-Programme, Variablen und Maschinencodeprogramme zur Verfügung. Der folgende Befehl zeigt die Adressen und die Größe des zur Verfügung stehenden Speichers an:

PRINT PAGE, HIMEM (HIMEM-PAGE)
"FREIE BYTES"

Die Tabelle zeigt eine kurze Beschreibung der verschiedenen Speicherbereiche des Acorn B: Für die meisten mit BASIC-Programmen eingesetzten Routinen eignet sich der Bereich für BASIC-Programmtexte am besten.

Speicherbereiche, die für Maschinencodeprogramme geeignet sind	Dateisystem	
	Cass.	Disk.
&0D00 — &0DFF Seite &D — nur wenn kein „Paged ROM" eingesetzt ist	(✓)	X
&0B00 — &0CFF Seite &B und &C — nur wenn keine vom Anwender definierten Zeichen oder Funktionstasten eingesetzt werden	(✓)	(✓)
&0A00 — &0AFF Seite &A — wenn die serielle Schnittstelle nicht angeschlossen ist	(✓)	(✓)
Mit DIM im BASIC-Programmbereich reservierter Platz		

✓ = OK, (✓) = OK mit Einschränkung, X ungenutzt

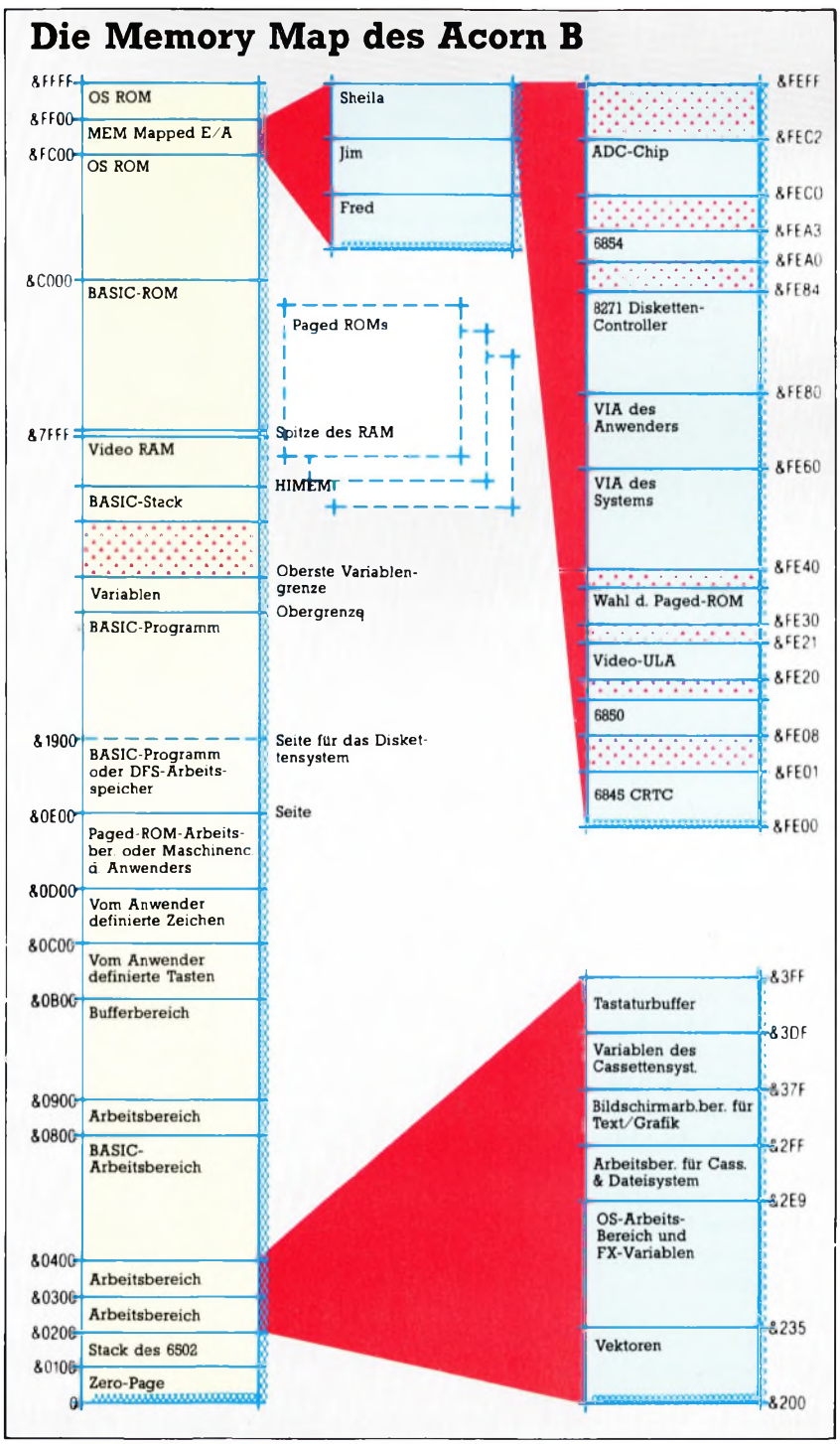
Maschinencode braucht für seine Abläufe normalerweise nur wenig Speicherplatz. Die meisten Maschinencodeprogramme des 6502 setzen dafür die Seite 0 (Zero-Page) ein, da die indizierte Adressierung nur mit dieser Seite funktioniert. Obwohl der Acorn B die Zero-Page oft für sein OS einsetzt, wurden einige Bytes für die Maschinencodeprogrammierung reserviert. Eine Liste mit den Funktionen der einzelnen OS-Speicherstellen nutzt dem Programmierer nur wenig. Diese Adressen sollten außerdem nicht direkt angesprochen werden, wenn das Programm auch auf anderen Betriebssystemversionen laufen soll.

Es gibt zwei Hauptmethoden, die OS-Routinen des Acorn B einzusetzen: OSBYTE und OSWORD.

- Mit OSBYTE können Sie die Abläufe vieler Betriebssystemroutinen beeinflussen, indem Sie Steuercodes und Parameter in die A-, X- und Y-Register des 6502 laden. Von BASIC aus läßt sich OSBYTE über den Befehl *FX anspre-

chen. Auf *FX müssen dabei zwei oder drei Zahlen folgen: Die erste Zahl gibt den Steuercode an, der an das Register A übergeben wird, die zweite Zahl geht in das X-Register und die dritte in Y. Im Maschinencode wird OSBYTE über die Adresse &FFF4 angesprochen. Die folgenden beiden Versionen haben die gleiche Funktion:

BASIC	Assemblersprache
*FX4,1	LDA #4 LDX #1 JSR &FFF4



Der Wert des A-Registers definiert exakt die Aufgabe, die ein bestimmter Aufruf von OSBYTE ausführt. Unser Beispiel beeinflußt die Cursortasten. Der an X übergebene Parameter bestimmt, ob die Cursortasten ihre normale Editierfunktion beibehalten oder einen ASCII-Wert liefern.

Leider lassen sich nicht alle OS-Routinen mit OSBYTE ansprechen, da das Modul nur wenige Parameter aufnehmen kann. Wenn Sie den Inhalt des A-Registers (das dem OS mitteilt, welche Routine von OSBYTE eingesetzt werden soll) ignorieren, können Sie nur zwei Parameter an X und Y übergeben. Für längere Parameterlisten müssen Sie dann entsprechend OSWORD einsetzen.

● Mit OSWORD lassen sich komplexe Vorgänge wie Tongeneration, Diskettenzugriffe etc. steuern. Hier zeigt sich der Unterschied zwischen den beiden Modulen: OSBYTE gibt an, wie das OS bestimmte Aufgaben ausführt, während sich mit OSWORD Aufgaben tatsächlich ausführen lassen. OSWORD erhält seine Parameter von einem „Parameterblock", auf den die Routine kurz nach dem Einsprung mit den X- und Y-Registern des 6502 zeigt. Dieser Parameterblock liegt im RAM. Seine Größe und sein Aufbau hängen von der aufgerufenen Funktion ab. Der Inhalt des A-Registers bestimmt, welche Funktion von OSWORD vom OS ausgeführt wird. Nach der Vorbereitung der Register und des Parameterblocks wird OSWORD über die Adresse &FFF1 angesprochen. OSBYTE und OSWORD sind die beiden Hauptmethoden, das OS einzusetzen.

Doch auch andere OS-Routinen lassen sich von BASIC aus mit einem Stern gefolgt von einem Befehl ansprechen. Der * veranlaßt, daß der Befehl nicht über den BASIC-Interpreter läuft, sondern an eine OS-Routine mit dem Namen OSCLI (Operating System Command Line Interpreter) übergeben wird. Diese Routine nimmt den Befehl entgegen und ruft die entsprechenden OS-Routinen auf. In der Tabelle haben wir die „Sternbefehle" des Acorn B aufgeführt. Eingaben mit Fehlern oder Befehle ohne die richtige Parameterzahl erzeugen die Meldung „Bad Command".

Ganzzahlvariablen beachten

Außer mit dem „Command Line Interpreter" (CLI) können Befehle auch direkt an das Betriebssystem übergeben werden. OSCLI hat zwei Einsatzmöglichkeiten: Sie können die Befehle (siehe Tabelle) vom Maschinencode aus oder als BASIC-Stringvariablen an CLI übergeben. Beachten Sie, daß die Ganzzahlvariablen A%, X% und Y% ihre Werte direkt in die Register A, X und Y laden. X% (oder das X-Register) und Y% (oder das Y-Register) zeigen im Speicher auf die Position des Strings, der als Sternbefehl interpretiert und von OSCLI ausgeführt werden soll.

BASIC-Version

```
10 DIM C 100
20 OSCLI=&FFF7
30 INPUT"ENTER COMMAND
",A$
40 $C=A$
50 X%=C MOD 256
60 Y%=C DIV 256
70 CALL OSCLI
80 GOTO 30
```

BASIC + Assemblerversion

```
10 DIM C 100
20 OSCLI=&FFF7
30 FOR I%=0 TO 2 STEP 2
40 P%=&C00
50 [ OPT I%
60 .code    LDX #C MOD 256
70          LDY #C DIV 256
80          JSR OSCLI
90          RTS
100 ]:NEXT I%
110 INPUT"ENTER COMMAND ",A$
120 $C=A$
130 CALL code
140 GOTO 110
```

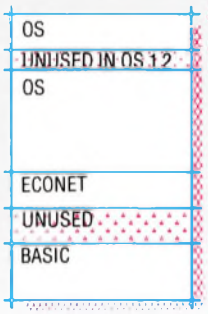
Dieses Programm erzeugt einen Prompt, wenn der Anwender einen Sternbefehl und Return eingibt. Mit dem DIM-Befehl in Zeile 10 reserviert das Programm 100 Bytes im Speicherbereich für BASIC-Variablen und initialisiert die Variable C mit der Anfangsadresse dieses Blocks. In die 100 Bytes können entweder Maschinencodeprogramme oder Daten gelegt werden. Der Befehl $C=A$ setzt die Bytes des in A$ gespeicherten Befehlsstrings in diesen Block von 100 Bytes (vom ersten reservierten Byte an aufwärts). In beiden Programmen werden die Register X und Y (die Variablen X% und Y%) geladen und dann die Steuerung an OSCLI übergeben. Damit wird der Befehlsstring ausgeführt.

Keine Programmunterbrechung

Dieses Modul läßt sich mit menügesteuerten Anwenderprogrammen einsetzen, wo es beispielsweise Disketten oder Cassetteninhalte ansprechen kann, ohne das laufende Programm zu unterbrechen. Dabei wird einfach der gewünschte Befehl in die Stringvariable geladen und zur Ausführung an OSCLI übergeben. Die Eingabe von *A$ funktioniert jedoch nicht. Das OS versucht dann einen Befehl namens *A$ zu finden, der nicht existiert.

Auch numerische Variablen lassen sich an Sternbefehle übergeben, wenn sie mit der Funktion STR$ zuvor in Strings umgewandelt wurden. Normalerweise nimmt das CLI keine Variablennamen an, sondern gibt die Fehlermeldung „Bad Command" aus.

Jeder vom OS nicht erkannte Sternbefehl wird an die Paged ROMs übergeben. Dabei wird jedes ROM „gefragt", ob es den Befehl identifizieren kann. Ist dies der Fall, wird er ausgeführt. Befehle, die auf diese Weise nicht zur Ausführung kommen, werden jedoch nur dann als „Bad Commands" behandelt, wenn kein schnelles Dateisystem (beispielsweise ein Diskettenlaufwerk) angeschlossen ist. In diesem Fall wird auch untersucht, ob auf der Diskette eine Datei mit dem angegebenen Befehlsnamen (allerdings ohne *) vorhanden ist. Wenn ja, wird die Datei in die Maschine geladen und als Maschinencodeprogramm interpretiert. Wird dagegen keine Datei gefunden, erscheint hier ebenfalls die Fehlermeldung „Bad Command".



Die Zero-Page des Acorn B
Der 6502 besitzt zwar nur eingeschränkte Indiziermöglichkeiten, kann aber die Zero-Page als Array von zusätzlichen Indexregistern einsetzen. Obwohl fast alle dieser Speicherstellen vom Betriebssystem belegt werden, stehen dem Programmierer einige Bytes zur Verfügung.

Die *-Befehle

	Beschreibung und Kommentar
*HELP	Gibt die Versionsnummer des BASIC an. Kann auch Informationen über Paged ROMs liefern – z. B. *HELP DFS
*BASIC	Ruft das BASIC-System auf. Eine Variante des Befehls – z. B. *WORD – ruft ein Paged ROM auf (hier View).
*CODE	Nur in OS 1.2. Damit kann man neue
*LINE	Befehle in das OS einsetzen.
*KEY	Für die Programmierung der roten Funktionstasten
*MOTOR n	Steuert Relais des Bandmotors: n=0 schaltet das Relais ab, n=1 schaltet es an
*ROM, *TAPE, *DISK, *NET	Initialisiert das Dateisystem – *TAPE aktiviert das Bandsyst. mit 1200 Baud und *DISK das Diskettensyst.
*FX	Gibt dem Programmierer die Möglichkeit, die Werte der OS-Variablen und damit die Abläufe des OS zu steuern
*RUN, *OPT, *LOAD, * *CAT, *SAVE	Dateisystembefehle, die in späteren Folgen untersucht werden
*SPOOL *EXEC	*SPOOL sendet Bildschirmausgaben an den Bildschirm und in eine Datei. *EXEC liest Daten aus einer Datei, als ob sie über die Tastatur eingegeben wären
*TV x,y	Beeinflußt die vertikale Anzeige des Bildschirms und der Bildschirmschnittstelle: x=0 löst keine Veränderung aus. x=1 bewegt die Bildschirmanzeige um eine Zeile nach unten, x=255 um eine Zeile nach oben. y=0 Interlace an, Y=1 Interlace aus.

Weitere Erklärungen dieser Befehle befinden sich im Handbuch des Acorn B.

Aktionsbereit

Diese Folge beschließt unsere Einführung in das Betriebssystem des Acorn B. Wir sehen uns den Einsatz der Vektoren genauer an und untersuchen, wie wir über das Betriebssystem mit dem Computer kommunizieren können.

Die meisten OS-Routinen (Betriebssystem-Routinen) des Acorn B arbeiten mit Vektoren. Das bedeutet, daß das OS für den Aufruf der Routine OSCLI zunächst eine Routine bei &FFF7 anspricht, die OSCLI (indirekte Weise) ansteuert. Die Routine findet dabei zunächst heraus, wo OSCLI überhaupt liegt, indem sie den Inhalt zweier Bytes auf Seite 2 des RAM untersucht. Diese zwei Bytes werden Vektor genannt: Das niederwertige Adreßbyte befindet sich im Vektorbyte mit der niedrigeren Bytenummer und das höherwertige Adreßbyte im höheren Vektorbyte. Bei der Routine OSCLI liegt das höherwertige Vektorbyte also in &209 und das niederwertige in &208. Diese Methode wird Hi-Lo-Adressierung genannt und für alle Adreßformate der 6502-Prozessoren eingesetzt. Bei jedem Reset legt das OS die Vektoren neu an. Diese etwas komplizierte Art, Betriebssystemroutinen aufzurufen, bietet große Vorteile.

Sie haben sicher bemerkt, daß alle bisher erwähnten OS-Routinen des Acorn B eine Adresse im Bereich von &FF00 und &FFFF aufrufen. Das hat gute Gründe: Für den Aufruf einer Adresse in diesem Bereich wird zunächst eine Routine angesprochen, die einen Sprung auf den Vektor dieser speziellen OS-Routine veranlaßt (wie beim Aufruf von CLI und OSCLI). Die aufgerufene Adresse zwischen &FF00 und &FFFF ist in allen OS-Versionen des Acorn B gleich. Falls die Routinen des OS-ROMs geändert werden müssen, haben die Konstrukteure der Maschine nur die entsprechenden Vektoradressen umzustellen. Wenn der Anwender die OS-Routinen über die korrekten Vektoren angesprochen hat, muß er bei Änderungen des OS seine Programme nicht verändern. In unterschiedlichen OS-Versionen kann daher der Inhalt der Vektoren durchaus variieren. Solange Sie die Einsprungsadressen zwischen &FF00 und &FFFF verwenden, werden Sie die Veränderungen jedoch nicht bemerken.

Der Einsatz von Vektoren hat noch einen weiteren Vorteil: Die Abläufe der OS-Routine lassen sich damit beeinflussen. Wenn Sie den Inhalt des Vektors verändern, können Sie statt der Maschinencodes des OS eigene Routinen verwenden. In einem späteren Teil des Kurses beschäftigen wir uns genauer mit den Vektoren der wichtigsten OS-Routinen.

Sehen wir uns einmal den Vektor USERV an, der die Speicherstellen &200 und &201 einnimmt. Das besondere an diesem Vektor ist, daß er im allgemeinen keine Funktion hat. Er wird von den *-Befehlen (*CODE und *LINE) aufgerufen. Bei ihrer Verwendung erscheint normalerweise die Fehlermeldung „BAD COMMAND".

Mit USERV können Sie jedoch eine Funktion definieren, die *CODE und *LINE ausführen sollen – sogenannte „anwenderdefinierte Befehle". *CODE eignet sich besonders für die Übergabe von Parametern an Maschinencodeprogramme:

Register	*CODE x,y	*LINE Textstring
A	0	1
X	Enthält den Wert des ersten Parameters nach *CODE. Er muß zwischen 0 und 255 liegen.	Enthält das niederwertige Adreßbyte, in dem sich das erste Stringzeichen nach *LINE befindet.
Y	Enthält den Wert des zweiten Parameters nach *CODE. Auch dieser Parameter muß zwischen 0 und 255 liegen.	Enthält das höherwertige Adreßbyte, in dem sich das erste Stringzeichen nach *LINE befindet.

Die Tabelle zeigt den Zustand der drei CPU-Register beim Aufruf der Routine, auf die der Inhalt von USERV zeigt. A enthält entweder Null oder Eins und zeigt damit an, welcher der beiden Befehle von USERV angesprochen wurde. Die Werte von X und Y hängen davon ab, ob Sie *CODE oder *LINE eingesetzt haben. *CODE 3,2 ruft daher die Routine auf, auf die USERV mit 0 in A, 3 in X und 2 in Y zeigt, wobei die beiden aufgeführten Parameter an diese Routine übergeben werden.

Einfacher *CODE-Befehl

Unser Beispielprogramm zeigt den Ablauf eines einfachen *CODE-Befehls. Die Maschinencoderoutine ist von Adresse &C00 an assembliert (das Ergebnis der Zuordnung in Zeile 40). Die Ganzzahlvariable P% wird dabei auf die gleiche Weise in den Befehlszähler (PC) geladen, wie A%, X% und Y% in die Regi-

Neu definierte Befehle (USERV)

```
10 USERV=&200
20 ?USERV=&00
25 ?(USERV+1)=&0C
30 FOR I%=0TO2STEP2
40 P%=&C00
50 [ OPT I%
60    CMP #0
70    BNE notcode
80    TXA
90   .loop
95    JSR &FFE3
100   DEY
110   CPY #0
120   BNE loop
130   RTS
140  .notcode
145   RTS
150 ]:NEXT I%
160
200 FOR rep=1 TO 10
210 FOR asc=33 TO 48
220 *CODE asc,rep
250 NEXT:NEXT
```

ster A, X und Y. Durch Setzen des niederwertigen Adreßbytes (&00) auf das Vektorbyte &200 und des höherwertigen Adreßbytes (&0C) auf das Vektorbyte &201, zeigt USERV auf diese Routine. Die Routine stellt die angegebene Zeichenfolge auf dem Bildschirm dar. Der erste Parameter von *CODE enthält dabei den ASCII-Code des Zeichens und der zweite die Angabe, wie oft das Zeichen wiederholt werden soll.

*LINE hat kein so breites Einsatzspektrum wie *CODE, wird aber auf die gleiche Weise eingesetzt. Ihre Hauptfunktion ist es, Textstrings an den Maschinencode zu übergeben. Diese beiden Befehle sind eine elegante Lösung, wenn nicht viele Parameter übergeben werden sollen.

Zeile 20 des Programms setzt USERV auf die Adresse der Maschinencoderoutine. Die Schleife zwischen Zeile 90 und 120 gibt das Zeichen, dessen ASCII-Code im A-Register liegt, Y-mal auf dem Bildschirm aus. Wird die Routine über *LINE angesprochen, fangen Zeile 60 und 70 diesen Fall ab und beenden die Routine. Zeile 200 bis 250 geben *CODE mit unterschiedlichen Parametern aus.

OSRDCH-Routine

Die Kommunikation mit einem Microcomputer wird hauptsächlich per Tastatur und Bildschirm abgewickelt. Wie steuert nun das OS diese beiden wichtigen Peripheriegeräte?

Wir untersuchen zunächst einen OS-Aufruf, mit dem wir Zeichen aus dem aktuellen Eingabekanal (Current Input Stream) lesen können. Die entsprechende Routine hat den Namen OSRDCH, liegt bei Adresse &FFE0 und wird über die Vektoren in &210 und &211 angesprochen. Sie empfängt über den Eingabekanal einzelne Zeichen. Wir werden uns daher zunächst ansehen, wie der Eingabekanal überhaupt gewählt wird. Es gibt zwei Haupteingabekanäle: die Tastatur und die RS423-Schnittstelle. Sie werden über OSBYTE oder *FX angewählt. Die folgende Tabelle zeigt diesen Befehl im Maschinencode und in BASIC:

Die Auswahl des Eingabekanals

n	Tastatur	RS423	Assembler	BASIC
0	✔	✘	LDA #2	*FX2,n
1	✘	✔	LDX #n	
2	✔	✔	JSR &FFF4	

*FX2,1 schaltet die Tastatur ab und aktiviert die RS423 als Eingabekanal. Die per RS423 empfangenen Daten werden nun wie Tastatureingaben behandelt. In der Assemblerversion führt der Wert 1 für die Variable n die gleiche Aufgabe aus: Der Rechner betrachtet die Daten via Schnittstelle als Tastatureingaben.

Der gewählte Eingabekanal läßt sich nun mit OSRDCH ansprechen. OSRDCH ist eigentlich nur für Assemblerprogramme nötig, da BASIC mit GET und INPUT schon über ausreichende Eingabemöglichkeiten verfügt. Nach dem Aufruf der Routine bei Adresse &FFE0 und dem Rücksprung befindet sich das vom Eingabekanal gelesene Zeichen im Register A. Wenn während des Lesens ein Fehler erkannt wurde, steht das Übertragsflag auf Eins, anderenfalls auf Null. Wenn also nach dem Rücksprung von OSRDCH das Übertragsflag C auf 1 steht, ist der in A gespeicherte Code nicht korrekt. Beim Einlesen per Tastatur löst das Drücken der Escape-Taste einen Fehler aus. Das A-Register enthält dann den Wert 27 (der ASCII-Wert für Escape). Dieser Fehler muß abgefangen werden, da das OS des Acorn B vom Programm eine Bestätigung erwartet. Doch zum Glück gibt es einen Weg.

Die Bestätigung geschieht durch den Aufruf von OSBYTE mit A=126. Damit werden verschiedene Teile des OS-Arbeitsbereiches gelöscht. Der BASIC-Interpreter erledigt diese Bestätigung nach jedem Escape natürlich automatisch. Die folgende Routine liest den Eingabekanal und reagiert erfreulicherweise auch auf ein Escape:

```
1000  .input  JSR &FFE0
1010          BCS error
1020          RTS
1030  .error  CMP#27
1040          BNE out
1050          LDA#126
1060          JSR&FFF4
1070  .out    RTS
```

Zeile 1000 ruft die Routine OSRDCH auf, und Zeile 1010 prüft das Übertragsflag. Ist es gelöscht, wird ein RTS zum aufrufenden Programm weitergeleitet – mit dem korrekt gelesenen Zeichen des A-Registers. Ist das Flag gesetzt, prüft Zeile 1030, ob der Fehler von einem Escape ausgelöst wurde. Falls ja, rufen Zeile 1050 und 1060 OSBYTE auf und bestätigen den Fehler. Auch für die String-Eingabe brauchen Sie keine eigene Assemblerroutinen zu entwickeln. Mit einer der OSWORD-Routinen lassen sich Strings vom Eingabekanal lesen. Wir werden OSWORD jetzt einsetzen, gehen aber später noch genauer darauf ein, um alle Unsicherheiten zu beseitigen.

Die OSWORD-Routinen liegen bei der Adresse &FFF1. Da es mehrere gibt, müssen wir vor dem Aufruf einen Wert in das A-Register stellen. Bei jedem Aufruf von OSWORD zeigen die Register X und Y auf einen Speicherblock („Control-Block"), der die Parameter für die Routine enthält. Das X-Register speichert dabei das niederwertige Byte der Control-Block-Adresse und das Y-Register das höherwertige. Die folgende Tabelle soll Ihnen den Aufbau verdeutlichen.

Der Control-Block von OSWORD	
Control-Block Eintrag	**Funktion**
0	Niederwertiges Adreßbyte, auf das die Zeichen geschrieben werden sollen
1	Höherwertiges Adreßbyte, auf das die Zeichen geschrieben werden sollen
2	Maximale Zeilenlänge
3	niedrigster akzeptabler ASCII-Wert
4	höchster akzeptabler ASCII-Wert

Bei der Zeicheneingabe über diese Routine behält die Delete-Taste ihre normale Funktion. Die Routine kann über Escape oder Return verlassen werden. Der Control-Block könnte nach einem Aufruf von OSWORD folgende Werte enthalten:

Beispiel für einen Control-Block von OSWORD	
Control-Block Eintrag	**Wert**
0	&00
1	&0C
2	&07
3	32
4	96

- Das erste Eingabezeichen ist in &C00 gespeichert, das zweite in &C01 etc.
- Es werden nur sieben Zeichen angenommen. Wenn Sie versuchen, mehr Zeichen einzulesen, wird ein akustisches Signal ausgegeben, und alle weiteren Zeichen werden ignoriert.
- Nur Zeichen mit ASCII-Codes zwischen 32 (Leerzeichen) und 96 (das $-Zeichen) werden angenommen, alle anderen zurückgewiesen.

Sie sehen, wie sich mit dieser Routine unerwünschte Zeichen ausfiltern lassen. Bei Beendigung der Routine gibt das C-Flag die Information aus, warum die Routine abgebrochen wurde. Bei C=1 wurde Escape gedrückt, bei C=0 die Return-Taste. Das Y-Register enthält die Länge des eingegebenen Strings zuzüglich des ASCII-Wertes für Return – falls die Zeichenkette damit beendet wurde. Beachten Sie, daß Sie diese Routine auf beiden Eingabekanälen verwenden können.

Die Eingabe von Daten in den Acorn B ist also recht einfach. Sehen wir uns nun an, wie Zeichen in den aktuellen Ausgabekanal (Current Output Stream) gesandt werden. Auch hier muß der Kanal über eine OS-Routine (der Befehl *FX3,n) ausgewählt werden. Jedes Bit von n steuert einen anderen Ausgabekanal an. So schaltet beispielsweise *FX3,1 die serielle Bildschirm- und Druckerausgabe an und ermöglicht die Ausgabe über SPOOL.

Parametertabelle zur Steuerung der Ausgabekanäle						
Bit	**0**	**1**	**2**	**3**	**4**	**6**
1	RS423 **AN**	Bildschirm **AUS**	Drucker **AUS**	Drucker **AN***	gespoolt **AUS**	Drucker **AUS****
0	**AUS**	**AN**	**AN**	**AUS***	**AN**	**AN****

*Drucker ist an oder aus, unabhängig davon, ob der Drucker auf andere Weise ausgeschaltet wurde.

**Drucker ist aus, wenn vor dem Zeichen kein CHR$(1) steht

Die Hauptroutine zum Senden von Zeichen an den Ausgabekanal heißt OSWRCH. Sie liegt bei &FFEE und wird über die Vektoren bei &20E und &20F aufgerufen. Ihr Einsatz ist einfach: Das A-Register muß nur mit dem ASCII-Code des Zeichens geladen und die Routine aufgerufen werden. Die folgenden drei Routinen stellen das Zeichen „A" auf dem Bildschirm dar:

```
1000 VDU 65
1000 PRINT (CHR$(65)
1000 LDA #65
1010 JSR &FFEE
```

Der BASIC-Befehl VDU hat eine ähnliche Wirkung wie OSWRCH. Mit Ausnahme des ASCII-Codes 127 (Delete) lassen sich alle Zeichen für die Codes zwischen 32 und 255 auf dem Bildschirm ausgeben. Die Zeichen im Bereich zwischen 0 und 31 haben jedoch Spezialfunktionen. Mit diesen Codes können wir per OSWRCH Grafik auf den Bildschirm bringen, die Befehle COLOUR und GCOL ausführen, Zeichen definieren oder den 6845-Chip steuern, der die Bildschirmanzeige des Acorn B kontrolliert.

Das Senden von Zeichen per OSWRCH wird oft auch als „Schreiben an die VDU-Treiber" bezeichnet. Die Tabelle der ASCII-Steuerzeichen zeigt, welche Wirkungen die Codes zwischen 0 und 31 haben, wenn sie an die VDU-Treiber gesandt werden. Es stehen damit alle Grafikfunktionen, die von BASIC aus möglich sind, auch im Maschinencode zur Verfügung.

Die OSWRCH-Routinen können alle normalen Grafikbefehle ansprechen. Unser zweites Programmbeispiel bringt eine rote Linie auf den Bildschirm. Die Zeilen 50 bis 75 setzen mit dem Befehl GCOL 0,1 die Farbe der Zeile auf Rot. Die Zeilen 90 bis 150 führen PLOT 5,100,100 aus (entspricht DRAW 100,100). Zeile 100 sendet den PLOT-Modus (in diesem Fall 5) an die VDU-Treiber, gefolgt von einer X-Koordinate (zwei Bytes im Lo-Hi-Format) und einer Y-Koordinate (zwei Bytes im Lo-Hi-Format). Wenn die 5 durch eine 4 ersetzt wird, läßt sich ein MOVE-Befehl ausführen (MOVE entspricht dem Befehl PLOT 4,x,y). Auch andere Grafikbefehle – wie PLOT 85 für das Zeichnen von Dreiecken – lassen sich auf diese Weise ausführen. Dabei ist zu beachten, daß die VDU-

Grafik mit OSWRCH

```
10 MODE 1
20 FOR I%=0TO2STEP2
30 P%=&C00
40 [OPT I%
50 LDA #18
55 JSR &FFEE
60 LDA #0
65 JSR &FFEE
70 LDA #1
75 JSR &FFEE
80
90 LDA #25
95 JSR &FFEE
100 LDA #5
105 JSR &FFEE
110 LDA #0
115 JSR &FFEE
120 LDA #100
125 JSR &FFEE
130 LDA #0
135 JSR &FFEE
140 LDA #100
145 JSR &FFEE
150 RTS
160 ]:NEXT I%
170 CALL &C00
```

Treiber fünf Bytes erwarten, wenn PLOT den Wert 25 als erstes Byte sendet. Werden diese Bytes nicht empfangen, können unerwünschte Effekte entstehen. Das trifft auf alle Vorgänge der VDU-Treiber zu, die mehr als ein Byte benötigen.

VDU23 ist ein weiterer Befehl für die VDU-Treiber. Mit ihm lassen sich neue Zeichen definieren.

```
VDU23,224,255,255,255,255,255,255,255,255
```

legt beispielsweise das (normalerweise nicht definierte) Zeichen 224 als Block an. Der Befehl gibt dem Anwender die Möglichkeit, die Zeichen 224 bis 255 in den Modi 0 bis 6 neu festzulegen. In Verbindung mit einer der OSBYTE-Routinen kann VDU23 auch Zeichen ändern. Alle VDU23-Aufrufe, die vom OS nicht erkannt werden (beispielsweise VDU23,0 . . .) werden über einen Spezialvektor bei &226 und &227 geleitet. Wenn Sie die Adresse dieses Vektors verändern, können Sie eigene VDU23-Routinen einsetzen.

Über VDU23 läßt sich auch der 6845-Video Steuerchip ansprechen. Der Befehl hat folgendes Format:

```
VDU23,0,Register,Wert,0,0,0,0,0,0
```

wobei „Register" das angesprochene Register des 6845 bezeichnet und „Wert" den Wert, der dorthin geschrieben werden soll. Das folgende Programm verändert mit VDU23 zwei der Register des 6845, die dem Chip mitteilten, welcher RAM-Bereich als Bildschirmspeicher eingesetzt werden soll. Das Programm legt den Anfang des Video-RAMs auf die Adresse &0000 und stellt den Arbeitsbereich des Acorn B auf den Bildschirm dar. Setzen Sie dort probeweise einmal einige Codezeilen, Array-Dimensionierungen, etc. ein. Die Routine ist zwar in BASIC geschrieben, läßt sich aber leicht in Assembler übersetzen.

```
10 MODE 0
20 VDU23,0,12,0,0,0,0,0,0,0
30 VDU23,0,13,0,0,0,0,0,0,0
40 VDU28,0,10,30,0:REM Textfenster anlegen
50 CLS
```

Zwei weitere OS-Aufrufe beziehen sich auf OSWRCH: OSNEWL und OSASCII. OSNEWL (bei &FFE7) bringt einen Zeilenvorschub und ein Return auf den Bildschirm, während jedoch OSASCII (bei &FFE3) eine Variante von OSWRCH ist und sich für die Textverarbeitung einsetzen läßt. Wenn über diese Routine das Zeichen 13 (Return) geschrieben wird, wird automatisch auch ein Zeilenvorschub (Zeichen 10) ausgegeben. Für an die VDU-Treiber gesandte Grafiktabelle dürfen Sie diese Routine nicht einsetzen, da ein zusätzliches CHR$(10) den Bildschirmaufbau beeinflussen kann.

Mit *SPOOL und *EXEC schließlich lassen sich Ein- und Ausgaben in das aktuelle Dateisystem stellen. *EXEC dateiname bewirkt, daß eine Datei mit dem angegebenen Dateinamen eröffnet und deren Inhalt eingelesen wird. *SPOOL schreibt Zeichen in die im Befehl angegebene Datei.

Damit endet unsere Einführung in das Betriebssystem des Acorn B. In den nächsten Folgen werden wir einige Maschinencoderoutinen untersuchen, mit denen sich die Bildschirmausgabe des Commodore 64 wesentlich verbessern läßt.

Tabelle der ASCII-Steuerzeichen

CODE	Zusatz-Byte	Beschreibung
0	0	Null
1	1	Nächstes Zeichen zum Drucker senden
2	0	Drucker anschalten
3	0	Drucker ausschalten
4	0	Text/Grafikcursor trennen
5	0	Text/Grafikcursor verbinden
6	0	VDU-Treiber einschalten
7	0	kurzes Klingelzeichen
8	0	Rückwärtsschritt des Cursors
9	0	Vorwärtsschritt des Cursors
10	0	Cursor eine Zeile nach unten
11	0	Cursor eine Zeile nach oben
12	0	Textbereich löschen
13	0	Return
14	0	Seitenschaltung ein
15	0	Seitenschaltung aus
16	0	Grafikbereich löschen
17	1	Textfarbe definieren
18	2	Grafikfarbe mit nächstem Byte definieren
19	5	logische Farbe definieren
20	0	logische Farbe auf Systemeinstellung stellen
21	0	VDU-Treiber abschalten
22	1	Darstellungmodus mit nächsten Byte auswählen. 22 und 7 schalten Modus 7 ein. HIMEM wird nicht verändert
23	9	dargestelltes Zeichen umdefinieren
24	8	Grafikfenster definieren
25	5	PLOT-Befehl ausführen
26	0	Text- und Grafikfenster zurückstellen
27	0	Null
28	4	Textfenster definieren
29	4	Grafikursprung definieren
30	0	Textcursor nach links oben stellen
31	2	Textcursor auf die Koordination x,y der nächsten zwei Bytes stellen. 31,10,10 setzt den Cursor beispielsweise auf die Position 10,10

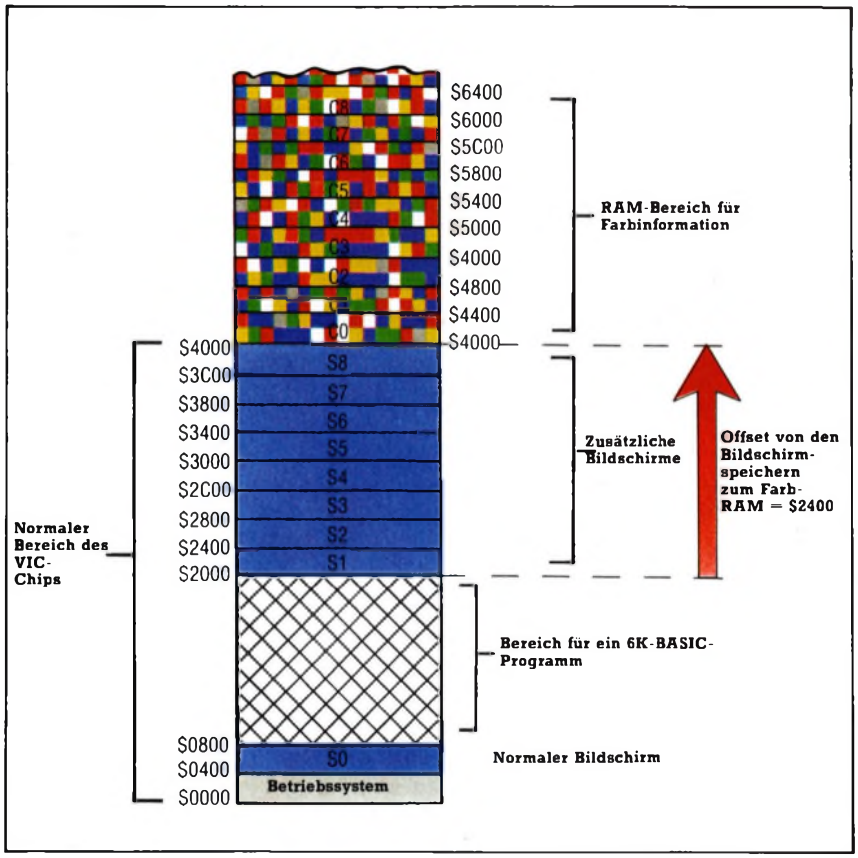
Bildfolgen

Der „Video Controller"-Chip (VIC) des Commodore 64 kann 16K RAM ansprechen. Normalerweise sind dies die ersten 16 K zwischen $ 0000 und $3FFF. Durch die Umstellung bestimmter Registerwerte des VIC können jedoch auch die anderen drei 16 K Blöcke zugänglich gemacht werden. Das Programm „Bildschirmwechsel" legt in den 16 K, die dem VIC normalerweise zur Verfügung stehen, „alternative" Bildschirmspeicher an. Die Farbinformationen jedes Bildschirms liegen direkt über diesem 16K-Bereich.

Auf dem Commodore läßt sich der Bildschirmspeicher problemlos in einen anderen RAM-Bereich verlegen. Wir sehen uns eine Maschinencoderoutine an, die bis zu acht Bildschirminhalte anlegen und zur Weiterverarbeitung speichern kann.

Auf dem Commodore 64 steuert der Spezialchip VIC II die Bildschirmanzeige und die Sprite-Verarbeitung. Dazu ruft der VIC-Chip Informationen aus verschiedenen Speicherbereichen ab, darunter vom Zeichen-ROM die Formen der Zeichen und vom Farb-RAM die Farbinformation. Der Bildschirmspeicher gibt an, welches Zeichen an den 1000 Bildschirmpositionen (25 Zeilen * 40 Spalten) dargestellt werden soll.

Beim Einschalten des Commodore 64 nimmt der VIC-II-Chip an, daß der Bildschirminhalt in



den 1000 Bytes gespeichert ist, die bei Adresse 1024 ($0400) anfangen. Seine erste Bildschirminformation holt er sich daher aus diesem Bereich. Durch eine Änderung eines Registerwertes im Inneren des VIC-Chips können wir jedoch veranlassen, daß der Chip auf einen anderen Bereich – gewöhnlich auf die ersten 16 KByte des Speichers – zugreift. Die oberen vier Bits des VIC-Steuerregisters bei 53272 ($D018) bestimmen, welcher Block von 1000 Bytes Länge als Bildschirmspeicher interpretiert werden soll. Die Tabelle zeigt die Bitwerte, die den 16 möglichen Bildschirmpositionen entsprechen.

Bitmuster	Anfang Bildschirminhalte	
0000XXXX	0	$0000
0001XXXX	1024	$0400 *
0010XXXX	2048	$0800
0011XXXX	3072	$0C00
0100XXXX	4096	$1000
0101XXXX	5120	$1400
0110XXXX	6144	$1800
0111XXXX	7168	$1C00
1000XXXX	8192	$2000
1001XXXX	9216	$2400
1010XXXX	10240	$2800
1011XXXX	11264	$2C00
1100XXXX	12288	$3000
1101XXXX	13312	$3400
1110XXXX	14336	$3800
1111XXXX	15360	$3C00

* Systemeinstellung

Für die Verlegung des Bildschirmspeichers an eine andere Stelle müssen wir die oberen vier Bits der Adresse 53272 ($D018) auf die Tabellenwerte setzen. Die unteren vier Bits dürfen nicht verändert werden (in der Tabelle durch XXXX gekennzeichnet), da sie eine andere Funktion steuern. Um auf die oberen vier Bits zugreifen zu können, ohne den Wert der unteren vier Bits zu verändern, müssen wir mit dem Inhalt des Registers ein AND mit 15 (binär 00001111) ausführen. Danach können wir mit OR den neuen Registerinhalt speichern. Für die Verlegung des Bildschirms in den letzten für den VIC zugänglichen Speicherbereich – Anfang bei 15360 ($3C00) – muß mit dem Inhalt des Registers ein OR mit 240 (binär 11110000) ausgeführt werden. Diese Aufgabe erledigt folgende BASIC-Zeile:

```
POKE 53272,(PEEK(53272) AND 15) OR 240
```

Bevor wir nun aber in den neuen Bildschirmspeicher schreiben können, müssen wir dem Betriebssystem des Commodore 64 mitteilen, daß sich die Position des Speichers verschoben hat. Das wird erreicht, indem wir das Hi-Byte der Anfangsadresse des neuen Speicherbereiches in die Adresse 648 ($0288) schrei-

ben. Als höchstmögliche Adresse muß hier $3C eingetragen werden (für den entsprechenden BASIC-Wert muß nur die Startadresse durch 256 geteilt werden).

Nachdem beide Register geändert wurden, können Sie wie gewohnt mit dem Speicher arbeiten. Es lohnt sich, für diesen Ablauf ein kurzes BASIC-Programm zu schreiben.

Durch die Verlegung des Bildschirmspeichers lassen sich interessante Effekte hervorrufen, da der Bildschirminhalt jetzt leicht ausgetauscht werden kann. Es muß allerdings auch das Farb-RAM versetzt werden, da es wichtige Informationen für den Bildschirmaufbau liefert. Nun lassen sich zwar mehrere Bildschirmbereiche einrichten, doch gibt es nur einen Bereich für Farbdaten, der sich auch nicht verlegen läßt. Wir müssen daher 1000 Bytes für die Farbinformationen reservieren. Beim Wechsel des Bildschirminhalts werden diese Bytewerte dann in das Farb-RAM kopiert und dessen bisheriger Inhalt gesichert.

Hauptaufgabe dieser Hilfsroutine ist es, den Speicher so zu organisieren, daß er zusätzliche Bildschirminhalte (mit ihren Farbdaten) aufnehmen kann und außerdem die Verschiebung der Speicherblöcke vornimmt. Da dem VIC-Chip 16 KBytes zugänglich sind, können wir mit unserem System bis zu acht Bildschirminhalte über ein BASIC-Programm steuern. Das Ablaufdiagramm zeigt, wie die Routine den Speicher anspricht.

Um sicherstellen zu können, daß kein Bildschirm oder Farbspeicher von BASIC-Programmen überschrieben wird, muß im Speicher die Obergrenze für BASIC-Programme mit folgendem Befehl heruntergesetzt werden:

POKE 55,0: POKE 56,32:CLR

Die Basisadresse jedes Bildschirminhalts kann nun mit dieser Zahl und der Formel

Bildschirmbasis = $1C00 + ($0400 * Nummer des Bildschirms)

berechnet werden. Für die Basisadresse des entsprechenden Farb-RAMs wird einfach ein Offset auf die Basisadresse des Bildschirmspeichers addiert. Die Formel lautet:

Farbbasis = $2400 + Bildschirmbasis

Neue Rahmenfarbe

Die Farbbereiche könnten zwar an jeder beliebigen Stelle des RAM liegen, doch bietet es sich an, sie direkt über den Bildschirmbereich zu setzen, den der VIC-Chip ansprechen kann. Beachten Sie, daß auch für den Bildschirm 0 ein Farbbereich abgestellt wird, dessen Farb-RAM ein unterschiedliches Offset hat.

Das VIC-Steuerregister und das Betriebssystemregister lassen sich auf folgende Weise einstellen:

VIC-Register = $70 + ($10 * Nummer des Bildschirms)
OS-Register = HI-Byte der Bildschirm-Basisadresse

Die Routine setzt aber nicht nur Register oder verschiebt Farbdaten, sondern legt auch die Farbe des Vordergrundes und des Rahmens fest. Diese beiden Parameter werden von einem Registerpaar des VIC gesteuert: 53280 ($D020) legt die Farbe des Rahmens fest und 53281 ($D021) die des Bildschirmhintergrundes. Unsere Routine richtet dafür eine eigene Tabelle ein.

Hilfreiches Flag

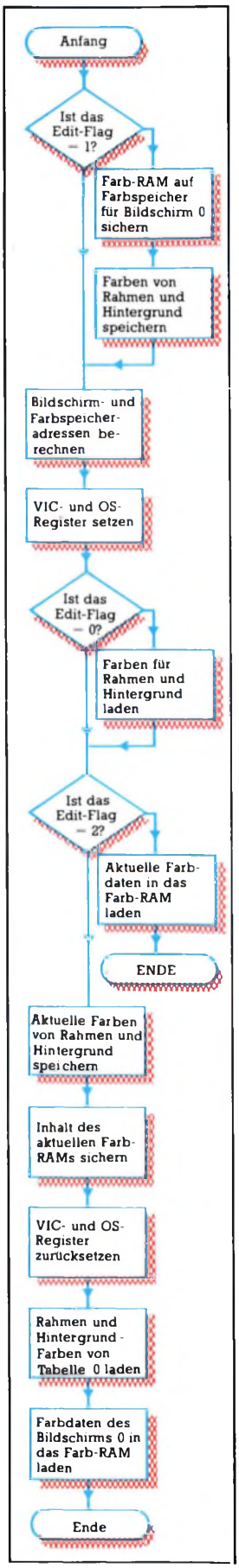
Die Routine hat zwei Betriebsarten: Sie kann einen bestimmten Bildschirminhalt entweder anzeigen oder editieren. In jedem Fall muß aber die Nummer des gewünschten Bildschirms mit POKE in 49152 ($C000) gesetzt werden. Um die Routine mit dem gleichen SYS-Aufruf ansprechen zu können, zeigt ein Flag an, welche Betriebsart gemeint ist. Das Flag wird durch ein POKE in die Adresse 49153 ($C001) gesetzt.

0 = anzeigen
1 = editieren

Der Editiermodus arbeitet mit dem normalen Bildschirmeditor und kann daher auch die Textfarben ändern, die Negativdarstellung setzen und den Bildschirm löschen. Dazu wird die Routine zuerst aufgerufen und das Edit-Flag auf Eins gesetzt. Nach Sicherung der normalen Farbdaten und der Hintergrund- und Rahmenfarben setzt die Routine die Register des VIC und des Betriebssystems und kehrt dann zum BASIC zurück. Das aufrufende BASIC-Programm übernimmt nun die Steuerung, stellt den Cursor in die linke obere Ecke und aktiviert den BASIC-Befehl INPUT. Dieser Befehl wartet, bis er ein Return (ASCII-Zeichen 13) erhält und ruft dann das restliche Programm auf. In der Zwischenzeit kann der Bildschirminhalt jedoch normal editiert werden.

Nach Beendigung des Editierens wird die Routine ein zweites Mal aufgerufen und das Edit-Flag auf 2 gesetzt. Dadurch werden vor dem Aufbau des normalen Bildschirms die Farbdaten und die Rahmen- und Hintergrundfarben gespeichert. Wenn nun die beiden letzten Parameter geändert werden sollen, müssen die entsprechenden Farbcodes mit POKE in die Speicherstellen 49154 ($C002) und 49155 ($C003) gesetzt werden.

Das BASIC-Programm stellt ein Menü dar, das die Wahl zwischen Anzeigen oder Editieren anbietet. Die Anzeigeroutine ruft auf Tastendruck jeden der acht unterschiedlichen Bildschirme nacheinander auf und wiederholt diesen Vorgang so lange, bis Return eingegeben wird. Das Programm stellt dann den normalen Bildschirminhalt wieder her und kehrt zum Menü zurück. Bei Wahl der Editieroption kann der Anwender für den gewählten Bildschirminhalt die Farben für Hintergrund und Rahmen angeben und dann mit dem Bildschirmeditor ein Bild erzeugen.



Ladeprogramm in BASIC

```
10 REM ************************
20 REM ** BASIC LOADER FOR  **
30 REM ** ALTERNATE SCREENS **
40 REM ************************
50 :
60 FORI=49152 TO 49439
65 READA:POKE I,A
70 CC=CC+A
75 NEXT
80 READ CS:IFCS<>CC THENPRINT"CHECKSUM ERROR":STOP
100 DATA255,255,0,0,191,255,0,0,255
110 DATA191,0,0,191,191,0,0,255,191,0
120 DATA0,255,191,0,0,255,255,0,173,1
130 DATA192,201,1,208,30,169,0,141,6
140 DATA192,169,64,141,7,192,32,224
150 DATA192,162,0,32,246,192,173,2,192
160 DATA141,32,208,173,3,192,141,33
170 DATA208,173,0,192,208,6,32,32,193
180 DATA76,139,192,169,0,141,4,192,141
190 DATA5,192,174,0,192,173,5,192,24
200 DATA105,4,202,208,250,24,105,28
210 DATA141,5,192,24,105,36,141,7,192
220 DATA173,0,192,162,4,10,202,208,252
230 DATA24,105,112,141,8,192,173,24
240 DATA208,41,15,13,8,192,141,24,208
250 DATA173,5,192,141,136,2,173,1,192
260 DATA208,6,174,0,192,32,211,192,201
270 DATA2,240,4,32,189,192,96,174,0
280 DATA192,32,246,192,32,224,192,32
290 DATA32,193,169,0,141,6,192,169,64
300 DATA141,7,192,162,0,32,211,192,32
310 DATA189,192,96,173,6,192,133,251
320 DATA173,7,192,133,252,169,0,133
330 DATA253,169,216,133,254,32,3,193
340 DATA96,189,9,192,141,32,208,189,18
350 DATA192,141,33,208,96,173,6,192
360 DATA133,253,173,7,192,133,254,169
370 DATA0,133,251,169,216,133,252,32,3
380 DATA193,96,173,32,208,157,9,192
390 DATA173,33,208,157,18,192,96,162,3
400 DATA160,0,177,251,145,253,136,208
410 DATA249,230,252,230,254,202,48,10
420 DATA208,240,177,251,145,253,160
430 DATA231,208,232,96
440 DATA36606:REM*CHECKSUM*
```

Bildschirmwechsel

```
3 REM  ********************************
5 REM  **                            **
10 REM **    ALTERNATE SCREENS       **
11 REM **                            **
12 REM ********************************
13 :
15 ON=8:REM FOR CASS ON=1
20 IFA=0 THEN A=1:LOAD"ALT SCREENS.HEX",ON,1
25 POKE55,0:POKE56,32:CLR:REM LOWER MEMTOP
40 SCNUMB=49152:  REM SCREEN NUMBER
70 EDITFG=49153:  REM 0=DISPLAY SCREEN
75 :              REM 1=EDIT SCREEN
77 :              REM 2=COPY C.RAM
80 ALT=49179:     REM M/C START ADDRESS
85 BRDCOL=49154:  REM BORDER COLOUR
87 PAPCOL=49155:  REM PAPER COLOUR
90 :
95 REM **** MAIN MENU ****
96 :
100 PRINTCHR$(147):REM CLEAR SCREEN
105 PRINTCHR$(154):REM LT BLUE LETTERS
110 ON$="■":REM 0 = CURSOR DOWN CHARS
130 PRINTTAB(8);ON$;"CBM 64 ALTERNATE SCREENS"
135 PRINTTAB(8);"----------------------"
140 PRINTTAB(5);ON$;"F1 - EDIT PICTURE"
160 PRINTTAB(5);ON$;"F3 - DISPLAY PICTURE SEQUENCE"
200 :
210 GETA$:IFA$=""THEN210:REM AWAIT KEYPRESS
220 IFA$="■" THEN GOSUB1000
230 IFA$="■" THEN GOSUB 1500
260 GOTO100
999 :
1000 REM **** EDIT SCREEN ****
1002 :
1005 EF=1:REM SET EDIT MODE
1010 PRINTCHR$(147)
1020 PRINTTAB(16);ON$;"EDIT MODE"
1030 PRINTON$;:INPUT"SCREEN NUMBER";SN$
1050 IFASC(SN$)<48 OR ASC(SN$)>56 THEN 1030
1060 PRINTON$;:INPUT"BORDER COLOUR";BC$
1070 IFVAL(BC$)=0 AND BC$()"0"THEN 1060
1080 PRINTON$;:INPUT"PAPER COLOUR";PC$
1090 IFVAL(PC$)=0 AND PC$()"0"THEN 1080
1100 :
1110 POKEEDITFG,EF
1120 POKESCNUMB,VAL(SN$)
1130 POKEBRDCOL,VAL(BC$)
1140 POKEPAPCOL,VAL(PC$)
1150 SYS ALT
1155 REM ** WAIT FOR RETURN **
1162 INPUT"■";X$:  REM S =HOME CURSOR
1165 REM ** SAVE SCREEN **
1170 EF=2
1175 POKE EDITFG,EF
1180 SYS ALT
1185 RETURN
1190 :
1500 REM **** DISPLAY SCREEN ****
1510 EF=0
1520 PRINT CHR$(147)
1545 SN=1
1550 POKE EDITFG,EF
1555 POKE SCNUMB,SN
1560 SYS ALT
1570 GET X$:IFX$=""THEN1570:REM AWAIT KEYPRESS
1572 IFX$=CHR$(13)THEN 1580:REM NORMAL SCREEN
1575 SN=SN+1:IFSN<9 THEN 1555
1577 SN=1:GOTO1555
1580 POKESCNUMB,0:SYS ALT
1600 RETURN
```

Commodore 64

```
;**********************
;**********************
;**                    **
;**     ALTERNATE      **
;**      SCREENS       **
;**        FOR         **
;**      CBM  64       **
;**                    **
;**********************
;**********************
;
FROM    =$FB            ;0 PAGE
TO      =$FD            ;POINTERS
;
CRAMLO=$00              ;START OF COLOUR RAM
CRAMHI=$D8
NCOLLO=$00              ;NORMAL COLOUR
NCOLHI=$40              ;BASE ADDRESS
NSCRHI=$04              ;NORMAL SCREEN BASE ADDRESS
NVCPOK=$10              ;NORMAL VIC REG VALUE
;
BLOCKS=$03              ;NO OF 256 BYTE BLOCKS
EXTRA  =$E7             ;EXTRA AMOUNT TO 1000 BYTES
;
COLOFF=$24              ;COLOUR OFFSET HIBYTE
SCROFF=$1C              ;SCREEN OFFSET HIBYTE
VICOFF=$70              ;VIC CTRL REG OFFSET
VCMASK=$0F              ;VIC CTRL REG LOBYTE MASK
VICREG=$D018            ;SCREEN LOCATION CONTROL REG
EDREG=$0288             ;SCREEN EDITOR KERNAL REG
BORDER=$D020            ;BORDER COLOUR REG
PAPER  =$D021           ;BACKGROUND COLOUR REG
;
*=$C000                 ;SET LOAD POINTER
;
SCNUMB  *=*+1           ;SCREEN NUMBER
EDITFG  *=*+1           ;EDIT MODE FLAG
BRDCOL  *=*+1           ;BORDER COLOUR
PAPCOL  *=*+1           ;PAPER COLOUR
;
SCBASE  *=*+2           ;SCREEN BASE STORAGE
CLBASE  *=*+2           ;COLOUR BASE STORAGE
VPOKE   *=*+1           ;TEMP STORE FOR VIC NUMBER
BRDTAB  *=*+9           ;BORDER COLOURS TABLE
PAPTAB  *=*+9           ;PAPER COLOURS TABLE
;
;**** SAVE SCREEN 0 ****
        LDA EDITFG
        CMP #$01        ;IF 0 OR 2
        BNE CALC        ;THEN DON'T SAVE
;
        LDA #NCOLLO
        STA CLBASE
        LDA #NCOLHI
        STA CLBASE+1
        JSR SAVE        ;SAVE CRAM TO C0
;
        LDX #$00
        JSR SAVEBP      ;SAVE B/P REGS
;
        LDA BRDCOL
        STA BORDER      ;SET NEW B/P
        LDA PAPCOL      ;COLOURS
        STA PAPER
;
;**** CALCULATE COLOUR BASE ****
;
CALC
        LDA SCNUMB
        BNE NOZERO
        JSR RESET       ;SET NORMAL REGS
        JMP TESTFG
;
NOZERO
        LDA #$00
        STA SCBASE
        STA SCBASE+1    ;INIT SCBASE
;
        LDX SCNUMB      ;LOAD SCREEN NUMBER
        LDA SCBASE+1    ;HI BYTE ONLY
MULT
        CLC
        ADC #$04
        DEX
        BNE MULT
;
        CLC
        ADC #SCROFF     ;ADD SCREEN OFFSET
        STA SCBASE+1
;
        CLC
        ADC #COLOFF     ;ADD COLOUR OFFSET
        STA CLBASE+1
;
;**** SET VIC AND EDITOR REGISTERS ****
        LDA SCNUMB
        LDX #$04
MORE
        ASL A
        DEX
        BNE MORE        ;MULT BY 16
;
        CLC
        ADC #VICOFF     ;ADD OFFSET
        STA VPOKE
        LDA VICREG
        AND #VCMASK
        ORA VPOKE
        STA VICREG      ;SET VIC REG
;
        LDA SCBASE+1
        STA EDREG       ;SET EDITOR REG
;
;**** TEST STATUS OF EDIT FLAG ****
TESTFG
        LDA EDITFG
        BNE NOLOAD      ;IF 0 DISPLAY MODE
        LDX SCNUMB
        JSR LOADBP      ;LOAD B/P TO REGS
;
;
NOLOAD
        CMP #$02
        BEQ CONT        ;IF 2 THEN SAVE RAM
;
        JSR LOAD        ;LOAD COL TO CRAM
        RTS
;
CONT
        LDX SCNUMB
        JSR SAVEBP      ;SAVE BP REGS
        JSR SAVE        ;SAVE CRAM TO COL
        JSR RESET       ;SET NORMAL REGS
        LDA #NCOLLO
        STA CLBASE      ;LOAD CBASE WITH C0 BASE
        LDA #NCOLHI
        STA CLBASE+1
        LDX #$00
        JSR LOADBP      ;RELOAD ORIG BORD/PAP COLS
        JSR LOAD        ;LOAD NRML SCR COLS
        RTS
;
;***** TRANSFER TO RAM S/R ****
LOAD
        LDA CLBASE
        STA FROM        ;LOAD 0 PAGE PTRS
        LDA CLBASE+1
        STA FROM+1
;
        LDA #CRAMLO
        STA TO
        LDA #CRAMHI
        STA TO+1
;
        JSR COPY        ;COPY RAM AREA
        RTS
;
;**** LOAD BORDER & PAPER COLS S/R ****
LOADBP
        LDA BRDTAB,X
        STA BORDER
        LDA PAPTAB,X
        STA PAPER
        RTS
;
;**** TRANSFER FROM RAM S/R ****
SAVE
        LDA CLBASE
        STA TO          ;LOAD 0 PAGE PTRS
        LDA CLBASE+1
        STA TO+1
;
        LDA #CRAMLO
        STA FROM
        LDA #CRAMHI
        STA FROM+1
;
        JSR COPY        ;COPY RAM AREA
        RTS
;
;**** SAVE BORDER & PAPER COLS S/R ****
SAVEBP
        LDA BORDER
        STA BRDTAB,X
        LDA PAPER
        STA PAPTAB,X
        RTS
;
;**** COPY 1000 BYTES S/R ****
;
COPY
        LDX #BLOCKS
        LDY #$00
NEXT
        LDA (FROM),Y
        STA (TO),Y
        DEY
        BNE NEXT
NXBLOC  INC FROM+1
        INC TO+1
        DEX
        BMI FINISH
        BNE NEXT
        LDA (FROM),Y
        STA (TO),Y
        LDY #EXTRA      ;EXTRA BYTES
        BNE NEXT
FINISH
        RTS
;
;**** RESET VIC AND EDIT REGS S/R ****
RESET
        LDA #NCOLLO
        STA CLBASE
        LDA #NCOLHI
        STA CLBASE+1
        LDA VICREG
        AND #VCMASK
        ORA #NVCPOK
        STA VICREG      ;RESTORE VIC REG
        LDA #NSCRHI
        STA EDREG       ;RESTORE EDREG
        RTS
```

Weich gerollt

Viele Arcade-Spiele simulieren schnelle Bewegung durch einen „vorbeiziehenden" Hintergrund. Auch der Commodore 64 unterstützt ein „weiches Scrollen" in die vertikale und horizontale Richtung.

Das Scrollen von Bildschirmen läuft in drei Stufen ab. Zunächst wird jedes Byte des Bildschirmspeichers um eine Position abwärtsbewegt. Da der Bildschirmspeicher die Zeilen als sequentielle Daten speichert, scheint sich dadurch jedes Zeichen um eine Position nach links zu bewegen – mit Ausnahme der Spalte am linken Bildschirmrand.

Bei diesem Vorgang verschwindet das Zeichen links oben im Bildschirm, während ein anderes Zeichen an der linken unteren Bildschirmkante auftaucht.

In der zweiten Stufe wird die entsprechende Spalte des Bildschirmspeichers in die rechte Spalte des Schirms kopiert.

Wenn nun die Scroll-Register des VIC verändert werden, scheint der Bildschirminhalt pixelweise in den Schirm „hineinzuscrollen".

Der Video Controller Chip (VIC) des Commodore 64 kann den Bildschirm des Commodore in jede Richtung um bis zu acht Pixel verschieben. Die horizontale Bewegung wird von den drei niederwertigen Bits des VIC-Registers bei 53270 ($DO16) gesteuert. Wenn diese Bits nacheinander auf Werte von 7 bis 0 gesetzt werden, verschiebt sich der Bildschirminhalt jeweils um ein Pixel nach links.

POKE 53270, (PEEK(53270) AND 248) + P

erledigt diesen Vorgang von BASIC aus (P ist ein Wert von 0 bis 7).

Wenn dieser Vorgang in einer Maschinencoderoutine verwendet wird, die sämtliche Bildschirmdaten um eine Position nach links bewegt und auf der rechten Seite eine neue Spalte einfügt, entsteht ein „sanft" rollendes Bild. Damit die Daten sich nicht ruckartig bewegen, muß allerdings die Bildschirmbreite des Commodore 64 von 40 auf 38 Spalten gesetzt werden. Dafür wird Bit 3 des Registers für horizontales Scrollen mit folgendem BASIC-Befehl auf 0 gesetzt:

POKE 53270, (PEEK(53270) AND 247)

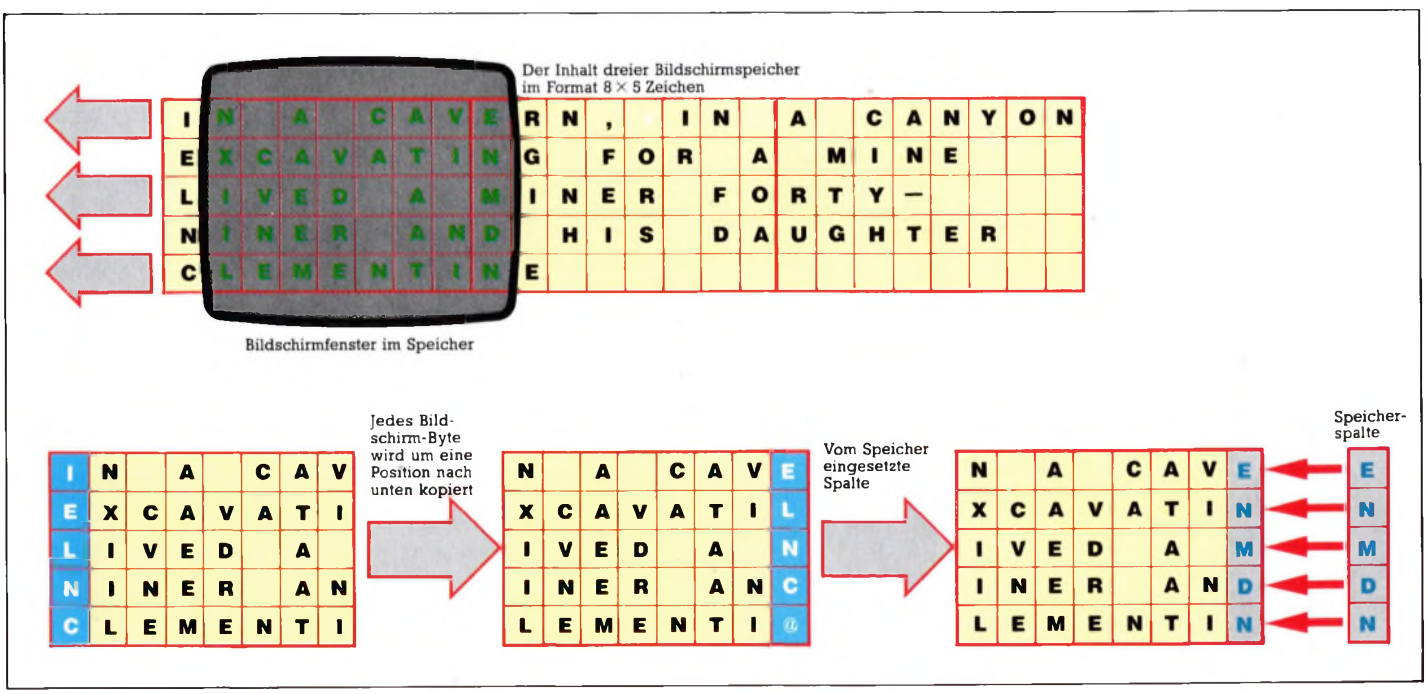
Wenn Bit 3 auf Eins steht, hat der Bildschirm 40 Spalten.

Das Diagramm zeigt, wie das weiche, horizontale Scrollen abläuft. Dabei müssen beim Bewegen oder Einsetzen von Bildschirmdaten auch jeweils die dazugehörigen Farbdaten verändert werden.

Der Vorgang ist eigentlich sehr einfach. Die Bildschirmdaten sind normalerweise in den 100 Bytes untergebracht, die bei Adresse 1024 ($0400) anfangen: Die ersten 40 Bytes stellen die oberste Zeile dar, die nächsten 40 Zeichen die zweite Zeile etc. Für die Illusion der Linksbewegung muß nur jedes Datenbyte in das darunterliegende Byte verschoben werden. Die Routine setzt dafür die Zeiger der Zero-Page und die indirekte Adressierung ein.

Wenn SB die Basisadresse des Bildschirmbereiches darstellt, dann ist das letzte Element der obersten Zeile SB+39, das der zweiten Zeile SB+79 etc. Damit die Bildschirmdaten im Speicher auf die gleiche Weise untergebracht sind, wie sie auf dem Schirm erscheinen (das heißt in Blöcken von 1000 Bytes), haben wir direkt daneben die Daten gespeichert, die wir einsetzen wollen (als das 41., 81. etc. Byte). Unsere Abbildung zeigt die Anordnung.

Der Zeiger, der auf den Anfang dieses Speicherbereichs zeigt, wird auf das erste Byte gesetzt, das auf dem Schirm erscheinen soll. Nach dem Erscheinen der ersten Spalte inkrementiert die Routine den Zeiger um eins, kopiert eine zweite Spalte auf die rechte Bildschirmseite und scrollt sie nach links. Wenn



dieser Vorgang vierzigmal wiederholt wird, wandern alle Daten einmal über den Bildschirm. Der Zeiger sollte dann um 960 (1000–40) inkrementiert werden, damit er den Anfang des nächsten Bildschirminhalts zeigt.

Dieser Ablauf muß auch für die Farbdaten ausgeführt werden. Zur Vereinfachung haben wir die Adressen der Farb-Bytes so angelegt, daß sie sich über ein konstantes Offset auf die Adressen ihrer entsprechenden Bildschirmbytes beziehen. Der Vorgang läßt sich für alle Bildschirminhalte wiederholen, die im Speicher angelegt wurden.

Um die Scroll-Routine einsetzen zu können, müssen jedoch mehrere Informationen eingegeben werden. Die Routine benötigt folgende Informationen:

1) die Anfangsadresse des Bereichs, in dem die Bildschirmdaten gespeichert sind,
2) das Offset in dem entsprechenden Farbbereich,
3) die Anzahl der Bildschirminhalte, die gescrollt werden sollen und
4) einen Verzögerungswert, der den Scroll-Vorgang verlangsamt.

Diese Daten können mit POKE in die Adressen gesetzt weden, die das Maschinencodeprogramm dafür reserviert hat.

Aufrufendes Programm in BASIC

```
10 REM ****************************
20 REM ****************************
30 REM **                        **
40 REM **  BASIC CALLING PROG    **
50 REM **  FOR SCROLL ROUTINE    **
60 REM **                        **
70 REM ****************************
80 REM ****************************
90 :
95 DN=8:REM FOR CASS DN=1
100 IFA=0THENA=1:LOAD"SCROLL.HEX",DN,1
110 POKE55,0:POKE56,32:CLR:REM LOWER MEMTOP
115 REMGOSUB1000:REM SET UP SIMPLE DISPLAY
120 :
130 LMEM  =49664:     REM START OF MEMORY
140 HMEM  =49665:     REM AREA
150 LCOFF =49666:     REM OFFSET TO COLOUR
160 HCOFF =49667:     REM MAP
170 NMSCR =49668:     REM NUMBER OF SCREENS
190 DELAY =49669:     REM DELAY VALUE
200 SCROLL=49670:     REM PROG START ADDRESS
210 :
220 REM PRINTCHR$(147):REM CLEAR SCREEN
230 INPUT"DECIMAL START ADDRESS";SA
240 HS=INT(SA/256):LS=SA-HS*256
250 POKELMEM,LS:POKEHMEM,HS
260 :
270 INPUT"NUMBER OF SCREENS";NS
290 POKE NMSCR,NS
300 :
310 INPUT"DECIMAL OFFSET TO COLOUR MAP";OS
320 HO=INT(OS/256):LO=OS-HO*256
330 POKELCOFF,LO:POKEHCOFF,HO
340 :
350 INPUT"DELAY VALUE < 256";DV
360 IF DV>255 OR DV<0 THEN 350
370 POKEDELAY,DV
380 :
390 SYS SCROLL
400 POKE53270,PEEK(53270)OR8
```

Das aufrufende Programm lädt den Maschinencode in den Speicher und holt sich über INPUT die notwendigen Informationen. Das Programm übersetzt die eingegebenen Daten – falls nötig – in das Format Lo-Byte/Hi-Byte und legt sie in den reservierten Speicherstellen ab. Danach wird die Maschinencoderoutine aufgerufen.

Es kann jede beliebige Anfangsadresse, jedes Offset oder jede Bildschirmzahl angegeben werden. Das Ergebnis hängt jedoch davon ab, ob an den angegebenen Positionen auch wirklich Bildschirmdaten untergebracht sind. Sie können die Routine leicht testen, wenn Sie ein kurzes BASIC-Programm laden und starten, das bei der Anfangsadresse 8192 zwei einfache Bildschirminhalte anlegt. Das Offset zu den Farbdaten ist 3000 Bytes. Für das Scrollen der Daten muß das aufrufende Programm folgende Informationen erhalten:

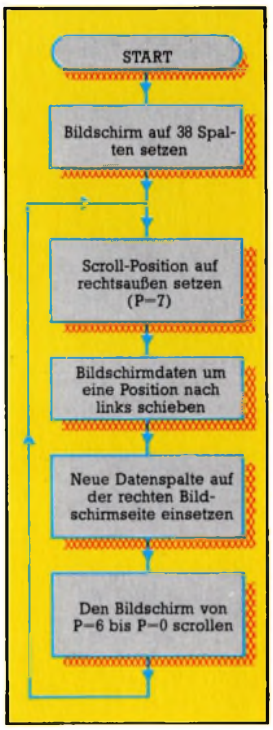
1) Anfangsadresse in Dezimal: 8192
2) Farb-Offset: 3000
3) Anzahl Bildschirme: 2
4) Verzögerung: 255

Ladeprogramm in BASIC

```
10 REM ************************
15 REM **  BASIC LOADER      **
20 REM ** FOR HORIZONTAL     **
30 REM ** SCROLL ROUTINE     **
40 REM ************************
50 :
60 FOR I=49670 TO 49945
70 READ A:POKE I,A
80 CC=CC+A
90 NEXT
92 READ CS:IF CS<>CC THEN PRINT"CHECKSUM ERROR":STOP
100 DATA173,22,208,41,247,141,22,208
110 DATA174,4,194,160,40,138,72,152,72
120 DATA173,22,208,41,248,24,105,7,141
130 DATA22,208,169,0,133,251,169,4,133
140 DATA252,169,0,133,253,169,216,133
150 DATA254,162,3,160,1,177,251,136
160 DATA145,251,200,177,253,136,145
170 DATA253,200,200,208,241,230,252
180 DATA230,254,177,251,198,252,136
190 DATA145,251,200,177,253,198,254
200 DATA136,145,253,230,252,230,254
210 DATA202,208,213,160,1,177,251,136
220 DATA145,251,200,177,253,136,145
230 DATA253,200,200,192,232,144,239
240 DATA173,0,194,133,253,173,1,194
250 DATA133,254,169,39,133,251,169,4
260 DATA133,252,32,241,194,173,0,194
270 DATA24,109,2,194,133,253,173,1,194
280 DATA109,3,194,133,254,169,39,133
290 DATA251,169,216,133,252,32,241,194
300 DATA162,6,173,22,208,41,248,141,22
310 DATA208,138,24,109,22,208,141,22
320 DATA208,172,5,194,136,208,253,202
330 DATA16,231,173,0,194,24,105,1,141
340 DATA0,194,173,1,194,105,0,141,1
350 DATA194,104,168,104,170,136,240,3
360 DATA76,19,194,173,0,194,24,105,192
370 DATA141,0,194,173,1,194,105,3,141
380 DATA1,194,202,240,3,76,17,194,96
390 DATA162,25,160,0,177,253,145,251
400 DATA202,240,29,165,251,24,105,40
410 DATA133,251,165,252,105,0,133,252
420 DATA165,253,24,105,40,133,253,165
430 DATA254,105,0,133,254,76,245,194
440 DATA96
450 DATA40227:REM*CHECKSUM*
```

Einstellroutine für die Anzeige

```
1000 REM **** SET UP DISPLAY ****
1010 CL=3000:REM OFFSET TO COLOUR MAP
1020 SS=8192:REM START OF DISPLAY MAP
1030 FORI=SS TO SS+479
1040 POKEI,1:REM SCREEN CODE FOR 'A'
1050 POKEI+CL,1:REM WHITE
1060 POKEI+480,2:REM SCREEN CODE FOR 'B'
1070 POKEI+CL+480,14:REM LIGHT BLUE
1080 NEXT
1085 FORI=SS+960TOSS+999
1090 POKEI,3:REM SCREEN CODE FOR 'C'
1100 POKEI+CL,3:REM CYAN
1110 NEXT
1999 :
2020 SS=9192:REM NEXT SCREEN START
2030 FORI=SS TO SS+479
2040 POKEI,3:REM SCREEN CODE FOR 'C'
2050 POKEI+CL,5:REM GREEN
2060 POKEI+480,4:REM SCREEN CODE FOR 'D'
2070 POKEI+CL+480,0:REM BLACK
2080 NEXT
2085 FORI=SS+960TOSS+999
2090 POKEI,5:REM SCREEN CODE FOR 'E'
2100 POKEI+CL,2:REM RED
2110 NEXT
```



Für eine weiche Scrollbewegung setzen wir auf dem Commodore 64 eine besondere Fähigkeit des Video Chips (VIC) ein. Mit dessen speziellen Scroll-Registern kann der Bildschirminhalt pixelweise über seine Grenzen hinausbewegt werden. Wenn diese Fähigkeit mit einer Kopierroutine im Maschinencode kombiniert wird, entsteht auf einem verkleinerten Bildschirm von 38 Spalten eine weiche Rollbewegung.

Horizontales Scrollen

```
;++++++++++++++++++++++++++
;++++++++++++++++++++++++++
;++                      ++
;++  HORIZONTAL SCROLL   ++
;++                      ++
;++     FOR CBM 64       ++
;++                      ++
;++++++++++++++++++++++++++
;++++++++++++++++++++++++++
;
SCRPTR=$FB              ;Ø PAGE
COLPTR=$FD              ;COPY POINTERS
;
MEMPTR=$FD              ;Ø PAGE PTR TO MEMORY
SCRLRG=$D016            ;HORIZ SCROLL REGISTER
SCRNLO=$ØØ              ;SCREEN START LOBYTE
SCRNHI=$Ø4              ;SCREEN START HIBYTE
COLRLO=$ØØ              ;COLOUR START LOBYTE
COLRHI=$D8              ;COLOUR START HIBYTE
BLOCKS=$Ø3              ;3*256 BYTE BLOCKS
EXTRA  =$E8             ;EXTRA BYTES TO 1ØØØ
NMCOLS=$28              ;NO OF COLUMNS
NMROWS=$19              ;NO OF ROWS
;
*=$C2ØØ
;
MEMLO   *=*+1           ;START OF MEMORY
MEMHI   *=*+1           ;TO BE SCROLLED
COFFLO  *=*+1           ;OFFSET TO COLOUR MAP
COFFHI  *=*+1
NMSCRN  *=*+1           ;NUMBER OF SCREENS
DELAY   *=*+1           ;DELAY LOOP VALUE
;
;++++ SET 38 COLUMN MODE ++++
;
        LDA SCRLRG
        AND #$F7
        STA SCRLRG
;
;++++ SET 1ST SCROLL POSITION ++++
;
        LDX NMSCRN
NEXSCR
        LDY #NMCOLS
START
        TXA
        PHA             ;PUSH X,Y REGS
        TYA             ;ONTO STACK
        PHA
;
        LDA SCRLRG
        AND #$F8
        CLC
        ADC #$Ø7
        STA SCRLRG
;
;++++ COPY SCRN & COLR ONE LEFT ++++
;
        LDA #SCRNLO
        STA SCRPTR      ;SET UP Ø PAGE
        LDA #SCRNHI
        STA SCRPTR+1    ;POINTERS FOR
        LDA #COLRLO
        STA COLPTR      ;COPY
        LDA #COLRHI
        STA COLPTR+1
;
        LDX #BLOCKS
AGAIN
        LDY #$Ø1
NEXT
        LDA (SCRPTR),Y
        DEY
        STA (SCRPTR),Y
        INY
        LDA (COLPTR),Y
        DEY
        STA (COLPTR),Y
        INY
        INY
        BNE NEXT
;
;++ COPY OVER PAGE BOUNDARY ++
        INC SCRPTR+1    ;INC HIBYTES
        INC COLPTR+1    ;OF Ø PAGE PTRS
        LDA (SCRPTR),Y
        DEC SCRPTR+1
        DEY
        STA (SCRPTR),Y;COPY OVER PAGE
        INY
        LDA (COLPTR),Y
        DEC COLPTR+1
        DEY
        STA (COLPTR),Y
        INC SCRPTR+1    ;INC Ø PAGE PTRS
        INC COLPTR+1    ;AGAIN
        DEX
        BNE AGAIN
;
;++ DO EXTRA BYTES ++
        LDY #$Ø1
ANTHER
        LDA (SCRPTR),Y
        DEY
        STA (SCRPTR),Y
        INY
        LDA (COLPTR),Y
        DEY
        STA (COLPTR),Y
        INY
        CPY #EXTRA
        BCC ANTHER      ;IF Y<EXTRA REPEAT
;
;++++ INSERT RIGHT COLUMN OF SCREEN ++++
;
        LDA MEMLO
        STA MEMPTR
        LDA MEMHI       ;SET UP Ø PAGE
        STA MEMPTR+1    ;PTRS TO MEMORY
        LDA #NMCOLS-1
        STA SCRPTR      ;SET UP Ø PAGE
        LDA #SCRNHI     ;PTRS TO SCREEN
        STA SCRPTR+1
        JSR COPY4Ø      ;COPY COLUMN
;
;++++ INSERT RIGHT COLUMN OF COLOUR ++++
;
        LDA MEMLO
        CLC
        ADC COFFLO      ;ADD OFFSET TO
        STA MEMPTR      ;COLOUR MAP
        LDA MEMHI       ;AND SET Ø PAGE
        ADC COFFHI      ;PTRS
        STA MEMPTR+1
        LDA #NMCOLS-1   ;SET UP Ø PAGE
        STA SCRPTR      ;PTRS TO COLOUR
        LDA #COLRHI     ;RAM
        STA SCRPTR+1
        JSR COPY4Ø      ;DO COPY
;
;++++ SCROLL POSITIONS 6 TO Ø ++++
;
        LDX #$Ø6
MORE1
        LDA SCRLRG
        AND #$F8
        STA SCRLRG
        TXA
        CLC
        ADC SCRLRG
        STA SCRLRG
;
        LDY DELAY       ;COUNT DOWN
MORE2
        DEY             ;DELAY VALUE
        BNE MORE2
        DEX
        BPL MORE1
;
;++++ INCREMENT MEMORY POINTER ++++
;
        LDA MEMLO
        CLC
        ADC #$Ø1
        STA MEMLO
        LDA MEMHI
        ADC #$ØØ
        STA MEMHI
;
;++++ CHECK FOR END OF MEMORY AREA ++++
;
        PLA
        TAY             ;GET X,Y REGS BACK
        PLA
        TAX             ;OFF STACK
        DEY
        BEQ NOJMP
        JMP START
NOJMP
        LDA MEMLO
        CLC
        ADC #$C8        ;ADD 1ØØØ-4Ø
        STA MEMLO       ;TO MEMORY POINTER
        LDA MEMHI
        ADC #$Ø3
        STA MEMHI
        DEX
        BEQ RETRN
        JMP NEXSCR
RETRN
        RTS
;
;++++ COPY EVERY 4ØTH BYTE S/R ++++
;
COPY4Ø
        LDX #NMROWS
        LDY #$ØØ
REPEAT
        LDA (MEMPTR),Y
        STA (SCRPTR),Y
        DEX
        BEQ FINISH      ;EVERY ROW DONE?
;
        LDA SCRPTR
        CLC
        ADC #NMCOLS     ;INC PTRS BY 4Ø
        STA SCRPTR
        LDA SCRPTR+1
        ADC #$ØØ
        STA SCRPTR+1
;
        LDA MEMPTR
        CLC
        ADC #NMCOLS
        STA MEMPTR
        LDA MEMPTR+1
        ADC #$ØØ
        STA MEMPTR+1
        JMP REPEAT
FINISH
        RTS
```

Der FX-Effekt

In dieser Folge untersuchen wir die OSBYTE-Aufrufe. OSBYTE ist eine einfache Methode, unterschiedliche Betriebssystemfunktionen des Acorn B einzusetzen.

Bei der ersten Untersuchung des Acorn B-Betriebssystems hatten wir kurz die OS-Aufrufe (OSBYTE) gestreift, die das Verhalten von OS-Funktionen verändern können. Mit *FX4,1 läßt sich beispielsweise die Reaktion des Acorn B auf die Betätigung der Cursortasten neu definieren.

In der 1.2-Version des OS gibt es über 100 dieser Aufrufe, die entsprechend viele OS-Funktionen des Acorn B steuern. Wir haben bereits festgestellt, daß durch den indirekten Aufruf von OS-Routinen die Programmfunktionen auch bei Änderungen der Hard- und Softwarekonfiguration nicht beeinflußt werden. OSBYTE ist eine praktische Methode, indirekt auf die OS-Routinen zuzugreifen.

Wenn Ihre Maschine mit der alten OS-Version 0.1 arbeitet, funktionieren einige Aufrufe von OSBYTE nicht, die im Anwenderhandbuch aufgeführt sind und die auch dieser Kurs behandelt. Wenn Sie *HELP <RETURN> eingeben, wird die Versionsnummer des Betriebssystems dargestellt.

Sehen wir uns zunächst an, wie OSBYTE von BASIC und vom Maschinencode aus eingesetzt wird. Wie die meisten OS-Aufrufe des Acorn B arbeitet auch OSBYTE mit Vektoren. Der OSBYTE-Vektor liegt bei den Adressen &20A und &20B. Die Tabelle zeigt, wie der OSBYTE-Aufruf *FX4,1 ausgegeben wird:

Der Einsatz von *FX von BASIC aus	Der Einsatz von USR von BASIC aus	Maschinencode
*FX4,1	A%=4:X%=1:D%=USR(&FFF4)	LDA #4 LDX #1 JSR &FFF4

Die Beispiele zeigen, daß alle OSBYTE-Routinen über die Adresse &FFF4 aufgerufen werden. Parameter lassen sich mit den Registern A, X und Y des Prozessors 6502 an OSBYTE übergeben. Wir können von BASIC aus A%, X% und Y% mit Werten belegen und dann über USR oder CALL eine Maschinencoderoutine aufrufen, die diese Werte in die Prozessorregister A, X und Y lädt. Mit USR (zweites Beispiel) läßt sich das Ergebnis eines Aufrufs auch wieder in eine BASIC-Variable übertragen. Dies ist besonders interessant, wenn die Daten in BASIC noch gebraucht werden. Wir kommen später noch darauf zurück.

Das dritte Beispiel ruft OSBYTE über ein kurzes Maschinencodeprogramm auf. Dabei werden die entsprechenden Register einfach mit Werten geladen und OSBYTE über &FFF4 angesprochen. Die Beispiele zeigen ebenfalls, wie sich die Parameter (4 und 1) auf drei verschiedene Weisen einfach an das OS übergeben lassen.

Unabhängig von der Art des OSBYTE-Aufrufs gibt das Register A immer an, welche der vielen OSBYTE-Routinen derzeit eingesetzt wird. X und Y übergeben die Parameter. Einige OSBYTE-Routinen brauchen keine Parameter, andere lediglich nur einen (Register X) und einige zwei (X und Y).

Hier ein paar Beispiele der drei Formate:

Keine Parameter	Ein Parameter	Zwei Parameter
*FX0	*FX4,1	*FX151,96,200

Beim Aufruf von OSBYTE-Routinen per *FX sind einige Punkte zu beachten. Wie alle *-Befehle kennt OSBYTE keine BASIC-Variablen. So löst die Ausführung des folgenden Codes die Fehlermeldung „Bad Command" aus:

```
a=4:b=1
*FX a,b
```

Das Problem läßt sich jedoch umgehen, wenn wir den FX-Befehl per OSCLI an das OS übergeben. So sieht es aus:

```
10 DIM C 100
20 a = 4
30 b = 1
40 $C = "*FX" + STR$a + "," +STR$b
50 X% = C MOD 256
60 Y% = C DIV 256
70 CALL &FFF7
```

Der FX-Befehl darf innerhalb einer Programmzeile nur als jeweils letzte Anweisung eingegeben werden:

```
*FX 4,1:PRINT "Hallo!"
```

erzeugt die Fehlermeldung „Bad Command", da das OS den Doppelpunkt und PRINT als Teil des FX-Befehls interpretiert.

Beide Probleme werden dadurch verursacht, daß *FX-Aufrufe nicht vom BASIC-Interpreter bearbeitet werden, sondern vom Befehlszeileninterpreter (CLI), der BASIC-Zeilen mit mehreren Befehlen nicht verarbeiten kann.

Ebenso wie sich mit X und Y Parameter an

OSBYTE übergeben lassen, so können auch die vom OS eingesetzten Variablen zurückgeben werden. Dieser Vorgang wird ebenfalls über USR oder eine Maschinencoderoutine gesteuert. Die Maschinencoderoutinen sind leichter zu bedienen, da die per USR gelieferten Werte erst decodiert (in Bits zerlegt) werden müssen. Die Register X und Y enthalten die rückgemeldeten Werte, wobei das Übertragsflag bei einigen Modulen gegebenenfalls auch vorhandene Fehler anzeigt.

CALLs von OSBYTE

Die zurückgegebenen Ergebnisse hängen natürlich von der eingesetzten Routine ab – das heißt von dem Wert, der sich bei Aufruf im A-Register befand. Nicht alle OSBYTE-Routinen geben Ergebnisse zurück, doch liefern viele Module wichtige Daten über das OS.

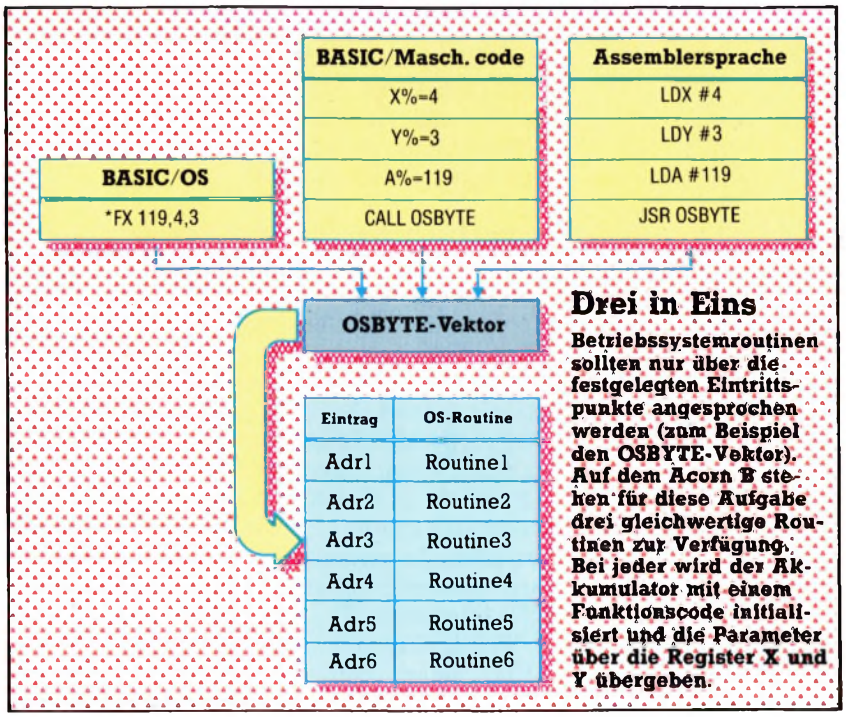
Zwei Arten von Informationen werden von OSBYTE zurückgemeldet: zunächst die Daten, die von einem Teil des Systems kommen – einer Schnittstelle oder einer Systemvariable. CALLs von OSBYTE, die diese Art von Information liefern, werden „read only"-CALLs genannt. Ein typisches Beispiel ist der OSBYTE-Aufruf mit A=129, den BASIC für die Funktion INKEY() einsetzt.

Mit den Registern X und Y läßt sich auch die vom OS verlangte Zeitverzögerung einstellen. Das X-Register speichert dabei das niederwertige Byte der Verzögerung (in Hundertstelsekunden) und Y das höherwertige. Für den Aufruf dieser Funktion können Sie die unten abgedruckte Maschinencodeversion einsetzen. Sie veranlaßt das System, bis zu einer Sekunde auf das Drücken einer Taste zu warten. Die Register X und Y geben die Werte zurück. Wenn das Übertragsflag auf 0 steht und das Y-Register ebenfalls 0 enthält, wurde der CALL mit einem Tastendruck beendet. Der ASCII-Wert der Taste ist dann in X gespeichert. Wenn Y den Wert 255 enthält und C auf 1 steht, wurde während der Verzögerungszeit keine Taste betätigt. Steht C auf 1 und Y auf 27, wurde die Escape-Taste gedrückt.

Das folgende Maschinencodebeispiel zeigt diesen OSBYTE-Aufruf. Wenn bei der Rückkehr das Übertragsflag gesetzt ist, verzweigt sie in eine weitere Subroutine, die dann über den Wert des Y-Registers prüfen könnte, ob eine alphanumerische Taste oder Escape gedrückt wurde.

```
1000 LDA    #129   /OSBYTE setzen
1010 LDX    #100   /Parameter
1020 LDY    #0
1030 JSR    &FFF4  /OSBYTE-Aufruf
                   ausführen
1040 BCS    Fehler
1050 RTS
1060 .Fehler       /Fehlerbehandlung
```

Es ist wichtig, sich diesen Aufbau einzuprägen, da er in ähnlicher Form auch für andere OS-BYTE-Aufrufe Verwendung finden kann. Besonders die Prüfroutinen sichern ein zuverlässiges Arbeiten und sollten stets zur Absicherung Verwendung finden.



Drei in Eins

Betriebssystemroutinen sollten nur über die festgelegten Eintrittspunkte angesprochen werden (zum Beispiel den OSBYTE-Vektor). Auf dem Acorn B stehen für diese Aufgabe drei gleichwertige Routinen zur Verfügung. Bei jeder wird der Akkumulator mit einem Funktionscode initialisiert und die Parameter über die Register X und Y übergeben.

Andere CALLs von OSBYTE, speziell die Aufrufe mit einem Wert zwischen 166 und 255, können sowohl Variablen aus dem OS lesen als auch dorthin schreiben. Wie kann nun OSBYTE überhaupt wissen, ob ein Lese- oder Schreibvorgang ausgelöst wurde?

Die Antwort ist einfach: Wenn an OSBYTE ein Parameter übergeben werden soll, muß das X-Register diesen Wert enthalten und Y auf 0 stehen. Für das Lesen einer der Systemvariablen wird X auf 0 und Y auf 255 gesetzt und dann der Aufruf durchgeführt. Die Ergebniswerte befinden sich danach in den Registern X und Y.

Wichtige Funktionen

OSBYTE-Aufrufe erledigen die „Verwaltung" des Betriebssystems und werden für viele Aufgaben eingesetzt – Diskettensystem, Tastatursteuerung, Econet, die Tasten Escape und Break etc. Es ist unmöglich, hier alle OSBYTE-Aufrufe aufzuführen. Wir werden uns daher auf einige wichtige Funktionen beschränken, die im Anwenderhandbuch des Acorn B nicht ausreichend erklärt sind.

Funktionstasten: *FX18 hat keine Parameter. Der Befehl löscht im Speicher die aktuellen Definitionen der Funktionstasten und macht so eine Neubelegung möglich.

*FX225 bis *FX228: Wenn die Funktionstasten nicht mit Strings programmiert sind, bewirkt dieser Aufruf, daß die roten Funktionstasten ASCII-Werte ausgeben. Zum Beispiel veranlaßt *FX225,n, daß die Funktionstaste f0 den ASCII-

Code für n liefert, f1 den ASCII-Code für (n+1) etc. *FS226 führt die gleiche Aufgabe für die mit Shift gedrückten Funktionstasten aus, *FX227 für die mit Control gedrückten Tasten und *FX228 für die gemeinsam mit Shift und Control betätigten Funktionstasten.

Video-Funktionen und VDU-Treiber: Der Bildschirm des Acorn B wird hauptsächlich mit Werten gesteuert, die an VDU-Treiber übermittelt werden. Etliche Aufrufe von OSBYTE sind dabei sehr hilfreich.

*FX19: Der Befehl arbeitet ohne Parameter und eignet sich für die Programmierung von Bewegungsgrafik. Er bewirkt, daß der Computer so lange wartet, bis das nächste Bild vollständig dargestellt ist. Der Aufruf sollte in jeder Pause eingesetzt werden. Da die Anzeige fünfzigmal pro Sekunde wiederholt wird, lassen sich damit Verzögerungen einsetzen oder Interupts an die CPU ausgeben.

*FX218: Dieser Schreib-/Leseaufruf von OSBYTE liefert die Länge der „VDU-Warteschlange". Wir hatten bereits erwähnt, daß auf einige VDU-Codes andere Codes folgen müssen. Die Anzahl der Bytes, auf die die VDU-Treiber zu einem gegebenen Zeitpunkt warten, ist die Zahl der Bytes in der VDU-Warteschlange. Der Befehl wird hauptsächlich als Lesevorgang eingesetzt. Das Ergebnis befindet sich dann erwartungsgemäß im X-Register.

*FX20: Mit diesem Aufruf können Sie alle ASCII-Zeichen zwischen 32 und 255 umdefinieren (und nicht nur eine begrenzte Anzahl). Im Modus 0 bis 6 sind dabei auch die normalen Tastaturzeichen zugänglich. Für die Umdefinierung von Zeichen, deren Codes zwischen 32 und 128 liegen, muß im Speicher Platz reserviert werden (der BASIC-Bereich wird mit PAGE auf einen höheren Wert gesetzt). Die Umdefinierung arbeitet mit VDU23 und Blöcken von 32 Zeichen, wobei jeder Block 256 Bytes benötigt. Der einzige Parameter, der benötigt wird, wird in das X-Register übergeben. Im Handbuch ist diese Technik ausführlich beschrieben.

X-Parameter	Angesprochener Buffer
0	Tastatur
3	Drucker
4	Tonkanal 0
5	Tonkanal 1
6	Tonkanal 2
7	Tonkanal 3

Tonerzeugung: *FX210,n schaltet sämtliche Soundeffekte ab, und *FX210,0 stellt sie wieder an. Setzt man einen anderen Parameter als X ein, so wird die Klangausgabe deaktiviert.

*FX211 bis *FX214: Diese als Schreib-/Leseaufrufe ausgelegten Routinen steuern den von CTRL-G oder VDU7 erzeugten Klang. *FX211,n bestimmt den Kanal des Tonerzeugungschips, auf dem der VDU7-Klang generiert wird. *FX212,n legt die Lautstärke der fertigen Töne fest oder definiert die zur Tonerzeugung notwendige Hüllkurve. Die Codierung der Lautstärke hat das gleiche Format wie der Amplitudenparameter im BASIC-Befehl SOUND (−15 maximale und 0 minimale Lautstärke, positive Werte bezeichnen die Nummer der Hüllkurve). Der per X-Register an den FX CALL übergebene Wert wird nach der Formel (n−1)*8 berechnet, wobei n der Parameter ist. *FX213,n steuert die Höhe des von VDU7 erzeugten Tones und *FX214,n die Dauer.

Escape-Taste: Mit *FX229,n läßt sich die Escape-Taste „abschalten". *FX229,1 schaltet sie aus, während *FX229,1 ihre normale Funktion wiederherstellt.

Mit *FX220,n kann der Anwender eine Taste definieren, die den Escape-Vorgang auslöst; die normale Escape-Taste wird dabei abgeschaltet. n bestimmt den ASCII-Wert der Taste. So veranlaßt *FX220,65, daß die Taste A eine Escape-Funktion auslöst.

Buffer: *FX21,n löscht die im Buffer vorhandenen Bytes. So beseitigt das Löschen des Tastaturbuffers alle Zeichen, die sich dort während eines Programmablaufs angesammelt haben. Wenn sich bei Ausführung eines GET-Befehls noch Zeichen im Tastaturbuffer befinden, holt GET sich ein Zeichen aus dem Buffer und nicht von der Tastatur. Das Löschen des Klangbuffers beendet die Tonerzeugung dieses Kanals, selbst wenn sich darin noch Befehle befinden, auf die der Klangchip wartet. X definiert den Buffer, der gelöscht werden soll (siehe Tabelle).

*FX138,n,m setzt den Wert m in den Buffer n ein. Die Buffernummern sind die gleichen wie bei *FX21. So legt *FX 138,0,65 das Zeichen A im Tastaturbuffer ab.

Bei all diesen Aufrufen von OSBYTE befinden sich die entsprechenden Parameter in den Registern A, X und Y.

Diese Aufstellung erhebt keinen Anspruch auf Vollständigkeit.

Wir haben nur einige wesentliche Funktionen von OSBYTE behandelt. Es gibt zwar noch weitaus mehr, doch verwenden alle die Techniken, die in diesem Artikel beschrieben sind.

Korrektur
Im Programm (Seite 1634) für anwenderdefinierte Funktionen (USERV) fehlte ein Teil. Hier das vollständige Listing.

```
5  DIM C 100
10 USERV=&200
20 ?USERV=&00
25 ?(USERV+1)=&0C
30 FOR I%=0 TO 2
   STEP 2
40 P%=&C00
50 [ OPT I%
60     CMP #0
70     BNE notcode
80     TXA
90    .loop
95     JSR &FFE3
100    DEY
110    CPY #0
120    BNE loop
130    RTS
140   .notcode
145    RTS
150 ]: NEXT I%
160 .
200 FOR rep=1 to 10
210 FOR asc=33 TO 48
220 asc$=STR$(asc)
230 rep$=STR$(rep)
240 code$="*CODE
    "+asc$+","+rep$
250 $C=code$
260 X%=C MOD 256
270 Y%=C DIV 256
280 CALL &FFF7
290 NEXT: NEXT
```

Der erste Block

In unserer Untersuchung des Acorn-Betriebssystems haben wir uns bisher mit den breitgefächerten Funktionen von OSBYTE beschäftigt. Wir untersuchen eine weitere Gruppe dieser Art – OSWORD.

Die bisher vorgestellten OSBYTE-Routinen haben einen wesentlichen Nachteil: Sie können nur zwei Parameter übergeben – einen im X-Register und einen in Y. Müssen nur wenige Parameter an das Betriebssystem übergeben werden, so reicht dies völlig aus, doch können die meisten Routinen nicht mit zwei Parametern gesteuert werden. Mit OSWORD lassen sich mehr als zwei Parameter einsetzen. Diese Modulgruppe ist eine praktische Alternative zu OSBYTE.

OSWORD-Routinen setzen einen Speicherbereich, den „Parameterblock" ein, um Parameter an das OS zu übergeben. Die Größe dieses Bereichs variiert je nach OSWORD-Routine und kann sich an jeder beliebigen Position des RAM befinden. Beim Aufruf von OSWORD teilt der Wert von Register A dem System mit, welche der Routinen angesprochen werden soll; die Register X und Y geben die Adresse des ersten Bytes des Parameterblocks an. X enthält dabei das niederwertige Adreßbyte und Y das höherwertige. Für einen Parameterblock bei der Adresse &0400 ist der Wert des X-Registers 00 und der von Y &0A:

Assembler	BASIC
LDX #0 LDY #&0A	X%=0 Y%=&0A

OSWORD wird über die Adresse &FFF1 aufgerufen und arbeitet wie OSBYTE mit Vektoren. Die Vektoren von OSWORD befinden sich bei Adresse &20C. Für den Aufruf der OSWORD-Routine mit Register A=1 müssen wir einen Parameterblock mit den entsprechenden Werten anlegen und die Routine dann mit folgendem Format aufrufen („parameter block" ist die Adresse des ersten Parameterblockbytes):

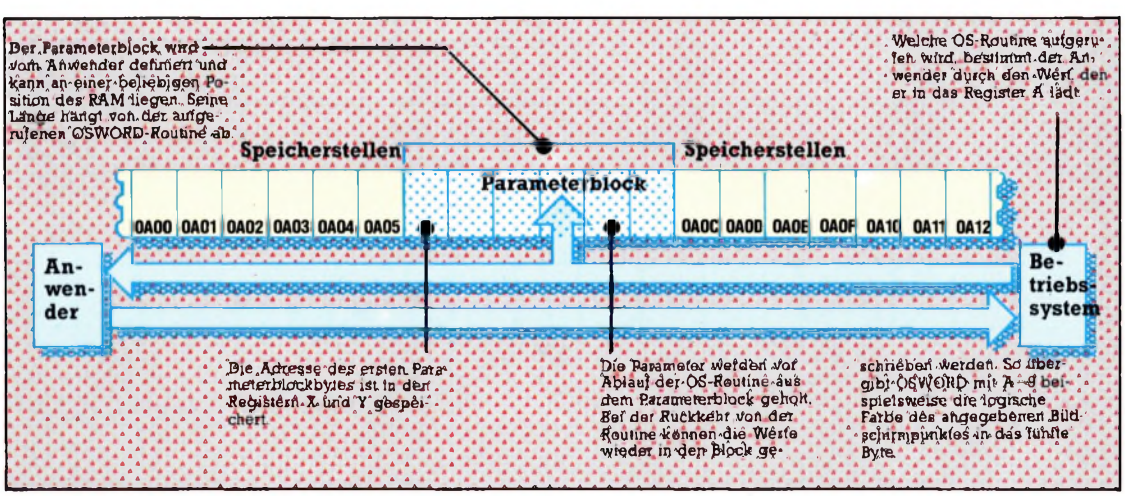
```
LDX  #parameter_block MOD 256
LDY  #parameter_block DIV 256
LDA  #1
JSR  &FFF1
```

Welche Werte in den Parameterblock geladen werden müssen, hängt natürlich von der angesprochenen OSWORD-Routine ab.

Die OSWORD-Routinen

Mit OSWORD-Routinen lassen sich Daten aus dem aktuellen Eingabekanal in den Speicher laden, die Befehle SOUND und ENVELOPE ausführen, das Diskettensystem ansprechen und viele weitere Funktionen steuern. Wir werden einige OSWORD-Calls genauer untersuchen. Die Routinen werden durch den Inhalt des A-Registers bezeichnet, das beim Aufruf die Routine eindeutig identifiziert (beispielsweise OSWORD mit A=0).

● **OSWORD mit A=0:** Diese Routine liest eine Zeile aus dem aktuellen Eingabekanal. Dabei kann ohne weiteres ganz einfach festgelegt werden, wieviele Zeichen das System maximal annehmen soll und welche niedrigsten und höchsten ASCII-Werte akzeptiert werden. Die ersten zwei Bytes des Parameterblocks geben die Speicheradresse an, bei der die Daten abgelegt werden. Der Parameterblock sieht dabei so aus:



OSWORD ist eine flexible Methode, OS-Routinen zu steuern, die mit mehr als zwei Parametern arbeiten.

Parameterblock von OSWORD mit A=0	
Byte	**Funktion**
0	Niederwertiges Byte der Bufferadresse
1	Höchstwertiges Byte der Bufferadresse
2	Maximale Anzahl der Zeichen
3	kleinstmöglicher ASCII-Code
4	größmöglicher ASCII-Code

Die Register X und Y zeigen auf Byte 0 des Blocks. Das folgende Programm gibt ein Beispiel des Aufrufs. Der Parameterblock legt dabei fest, daß immerhin bis zu sieben Zeichen mit ASCII-Codes zwischen 32 und 90 (inklusive) akzeptiert werden.

```
10 DIM C 20
20 A%=0
30 X%=C MOD 256
40 Y%=C DIV 256
50 C?0=&00
60 C?1=&0A: REM buffer at &0A00
70 C?2=7
80 C?3=32
90 C?4=90
100 CALL &FFF1
110 PRINT $(&0A00)
```

Starten Sie das Programm und geben Sie einige Zeichen ein. Delete funktioniert wie gewohnt, und Return oder Escape beenden die OSWORD-Routine. Zeichen mit ASCII-Codes außerhalb des angegebenen Bereiches werden zwar auf dem Bildschirm angezeigt, nicht aber in den Speicherbuffer übertragen. Werden zuviel Zeichen eingegeben, ertönt ein akustisches Signal. Bei Beendigung der Routine mit Escape setzt der Maschinencode das C-Flag, andernfalls steht C auf 0. Im Y-Register steht die Anzahl der eingegebenen Zeichen, wobei das Return mitgezählt wird.

● **OSWORDs mit A=1 und A=2:** Mit diesen Routinen kann der Maschinencodeprogrammierer auf die BASIC-Variable TIME zugreifen. OSWORD mit A=1 liest den aktuellen Wert von TIME. Die fünf Bytes dieses Wertes werden im Parameterblock gespeichert. Das Byte mit dem niedrigsten Wert ist in Byte 0 des Parameterblocks untergebracht und das höchstwertige Byte in Byte 5.

OSWORDs mit A=2 setzt TIME auf einen bestimmten Wert. Auch hier wird vor dem Aufruf das Byte mit dem niedrigsten Wert in Byte 0 des Parameterblocks geladen und das höchstwertige in Byte fünf. Das folgende Programm setzt den Wert von TIME auf 100:

```
10 DIM C 20
20 X%=C MOD 256
30 Y%=C DIV 256
40 A%=2
50 C?0=100
60 C?1=0
70 C?2=0
80 C?3=0
90 C?4=0
100 CALL &FFF1
```

● **OSWORDs mit A=3 und A=4:** Der Acorn B besitzt eine interne Uhr – den „Interval-Timer" –, die nicht von TIME eingesetzt wird, sondern von dem „Ereignissystem" (events system) des Computers (wir gehen später genauer darauf ein). Der Timer zählt von einem gesetzten Wert an in Hundertstelsekunden rückwärts und löst bei Null ein „Ereignis" aus. Das Ereignis läßt sich programmieren und kann nach der angegebenen Zeit im Computer Abläufe auslösen.

OSWORD mit A=3 liest den aktuellen Wert des Interval-Timers auf die gleiche Weise in den Parameterblock, in der OSWORD mit A=1 die Variable TIME speichert. Durch OSWORD mit A=4 kann der Timer auf einen bestimmten Wert gesetzt werden; hier gibt es Parallelen zu OSWORD mit A=2. Wenn ein Parameterblock den Wert &FFFFFFFC enthält, wird das Ereignis nach 0,004 Sekunden ausgelöst.

● **OSWORDs mit A=5 und A=6:** Diese beiden Spezialroutinen sprechen den zweiten Prozessor des Acorn B an. Der Zusatzprozessor kommuniziert über die Tube-Schnittstelle mit dem Acorn B und behandelt den Computer als E/A-Prozessor. Der Acorn B funktioniert nun als sogenannter „Sklave", der für den Zusatzprozessor alle zeitraubenden Arbeiten ausführt – beispielsweise die Abfrage von Tastatureingaben, die Steuerung des Bildschirmaufbaus und die allgemeinen Ein- und Ausgabefunktionen. Über die beiden OSWORD-Routinen kann der zweite Prozessor Speicherstellen im E/A-Prozessor ansprechen.

Parameterblock von OSWORD mit A=5 und A=6		
Byte	**OSWORD mit A=5**	**OSWORD mit A=6**
0	LSB der E/A-Prozessoradresse	LSB der E/A-Prozessoradresse
1	Adresse des zu lesenden Bytes	Adresse des zu schreibenden Bytes
2	Adresse des zu lesenden Bytes	Adresse des zu schreibenden Bytes
3	MSB des E/A-Prozessors	MSB des E/A-Prozessors
4	Nach dem Aufruf das Byte der oben angegebenen Adresse	Das Byte, das an die oben angegebene Adresse geschrieben wird

Die Tabelle zeigt den Aufbau des Parameterblocks für A=5 (ein Byte aus dem E/A-Speicher des Prozessors lesen) und für A=6 (in den E/A-Speicher des Prozessors schreiben). Wenn die Adresse das einfache 16-Bit-Format hat, wird ihr niederwertiges Byte bei (XY+0) abgelegt und das höherwertige bei (XY+1).

● **OSWORDs mit A=7 und A=8:** Über diese beiden Routinen läßt sich der Klang verändern (von Maschinencodeprogrammen und von BASIC aus). OSWORD mit A=7 simuliert den BASIC-Befehl SOUND, während OSWORD mit A=8 der ENVELOPE-Anweisung entspricht.

Hier die Aufrufe im einzelnen:

● **OSWORD mit A=7:** Die Routine benötigt einen Parameterblock von acht Bytes (die Tabelle zeigt die Belegung) und die Information, welcher Eintrag des Parameterblocks den Parametern des Befehls SOUND entspricht.

Parameterblock von OSWORD mit A=7	
Byte	**Funktion**
0	LSB der Kanalnummer
1	MSB der Kanalnummer
2	LSB der Amplitude
3	MSB der Amplitude
4	LSB der Tonhöhe
5	MSB der Tonhöhe
6	LSB der Dauer
7	MSB der Dauer

Das folgende Programm zeigt den Ablauf der Routine. Es simuliert SOUND 1,−15,100,30:

```
10  DIM C 20
20  X%=C MOD 256
30  Y%=C DIV 256
40  A%=7
50  C?0=1
60  C?1=0:REM Channel 1
70  C?2=-15 MOD 256
80  C?3=-15 DIV 256
90  C?4=100
100 C?5=0
110 C?6=30
115 C?7=0
120 CALL &FFF1
130 END
```

Da sich die gleiche Wirkung mit SOUND viel einfacher erzeugen läßt, hat diese Routine allerdings nur wenig Wert und wird daher auch wohl kaum Anwendung finden.

● **OSWORD mit A=8:** Der Parameterblock dieser Routine umfaßt vierzehn Bytes. Wie gewohnt zeigen dabei die Register X und Y auf den Block.

Der erste Blockeintrag gibt die Nummer von ENVELOPE an, die restlichen dreizehn Bytes enthalten die anderen ENVELOPE-Parameter. Auch hier läßt sich ENVELOPE vom BASIC aus leichter einsetzen.

Byte	Wert
0	2
1	4
2	0
3	0
4	0

Maschinencodeversion

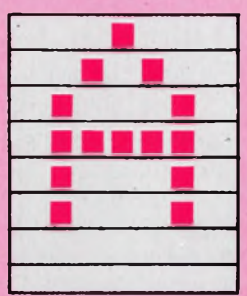
● **OSWORD mit A=9:** Die Routine ist die Maschinencodeversion des BASIC-Befehls POINT. Sie meldet die logische Farbe des angegebenen Pixels (oder den Wert 255, wenn der spezifizierte Punkt nicht aktiviert ist). Der Wert wird als fünftes Byte des Parameterblocks übergeben und hat folgenden Aufbau:

Parameterblock von OSWORD mit A=9	
Byte	**Funktion**
0	LSB der X-Koordinate
1	MSB der X-Koordinate
2	LSB der Y-Koordinate
3	MSB der Y-Koordinate
4	Logische Farbe

OSWORD mit A=10

Mit dieser praktischen Routine lassen sich vergrößerte Zeichen auf den Schirm bringen. Sie können damit die Zeichendefinitionen der Modi 0 bis 6 abrufen. Es wird ein Parameterblock von neun Bytes gebraucht, auf den wie gewöhnlich die X- und Y-Register zeigen. Das erste Byte des Blocks sollte den ASCII-Code des Zeichens enthalten.

Parameterblock von OSWORD mit A=10	
Byte	**Funktion**
0	gewünschtes Zeichen
1	erste Definitionszeile
2	zweite Zeile
3	dritte Zeile
4	vierte Zeile
5	fünfte Zeile
6	sechste Zeile
7	siebte Zeile
8	letzte Definitionszeile



Die in den Parameterblock zurückgegebenen Werte sind eine binäre Darstellung der Zeichendefinition.

● **OSWORD mit A=11:** Mit diesem Aufruf lassen sich die Farben untersuchen, die von VDU 19-Befehlen bestimmt wurden. Damit kann festgestellt werden, welche logische Farbe der dargestellten Farbe entspricht. Die Routine arbeitet mit einem Parameterblock von fünf Bytes, wobei das erste Byte den Wert der gewünschten logischen Farbe enthält. Hier ein Beispiel:

Nehmen wir an, daß der Befehl VDU 19,2,4,0,0,0 schon ohne weitere Vorbehalte ausgeführt wurde. OSWORD mit A=11 und dem Wert 2 im ersten Byte des Parameterblocks liefert dann entsprechend die Werte der nebenstehenden Tabelle.

● **OSWORD mit A=12:** Eine schnelle Version von VDU 19

● **OSWORD mit A=13:** Diese Routine liefert die X- und Y-Koordinaten der letzten beiden Punkte, die der Grafikcursor bei unserer Arbeit am Bildschirm angesprochen hat. Die Routine kann bei der Grafikprogrammierung sehr hilfreich sein.

Dateizugriff

Wir stellen das Dateisystem des Acorn B vor, das mit einer Reihe von Betriebssystemroutinen Daten an das Speichermedium sendet oder von dort empfängt. Hier untersuchen wir den Aufruf von OSFILE.

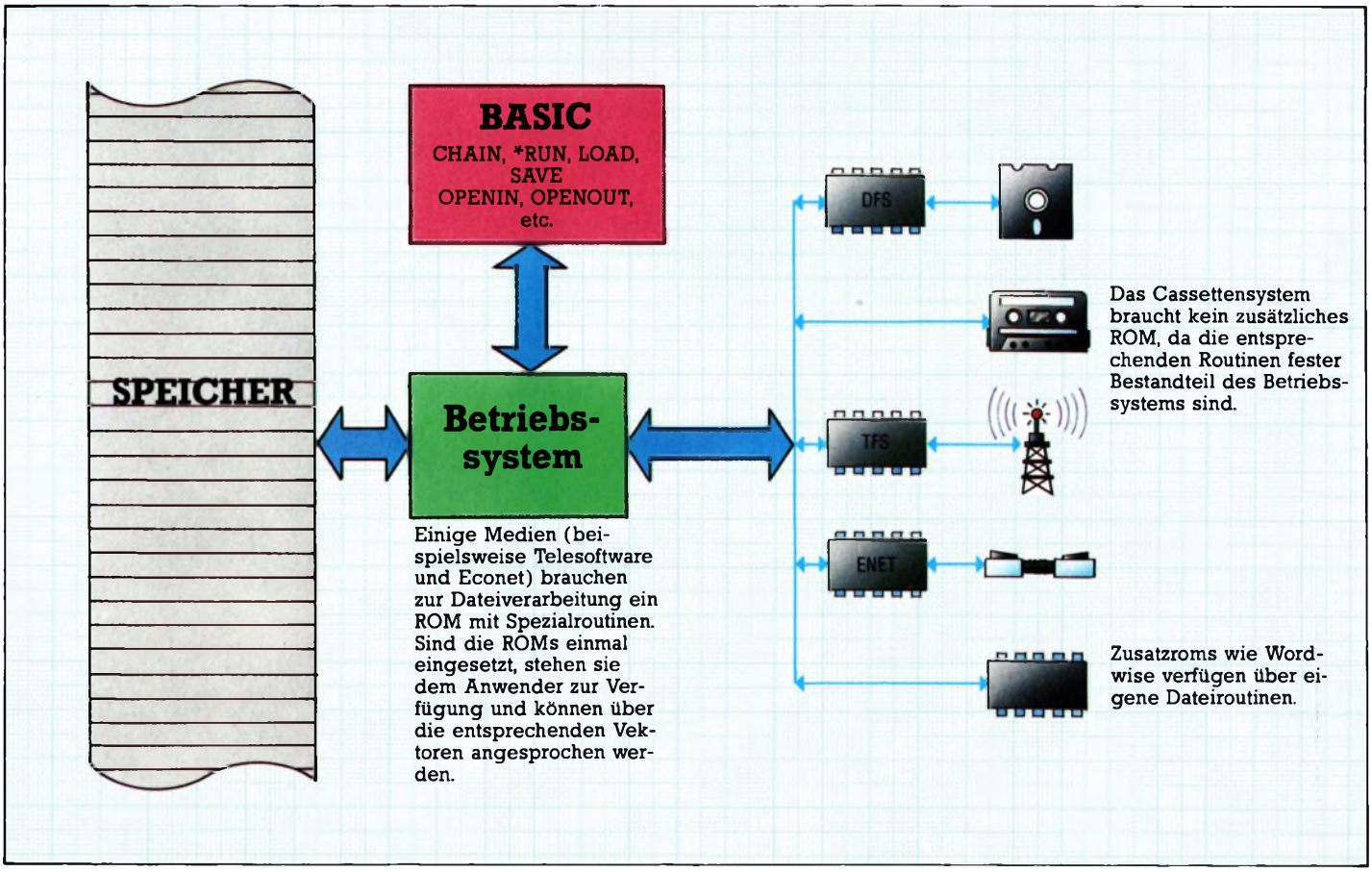
Dateisysteme enthalten eine Gruppe von Routinen, mit denen sich Daten zwischen dem Arbeitsspeicher, permanenten Speichern und anderen Medien (darunter auch Telesoftware und andere Netzwerke) übertragen lassen.

Das Dateisystem eines Computers steuert die Datenübertragung zwischen dem Arbeitsspeicher und anderen Speichermedien, beispielsweise einer Cassette oder Diskette. Der Acorn B kann neben diesen Speichermedien auch andere Dateisysteme – beispielsweise das Econet-Netzwerk und ROM-Cartridges – ansprechen. Obwohl eigentlich zu erwarten wäre, daß der Acorn B für jedes Gerät eine eigene Routine braucht, verwendet das Dateisystem die gleichen OS-Module für alle Geräte. Das heißt jedoch nicht, daß alle Geräte auf die gleiche Weise funktionieren. Viele Speichermedien brauchen ein eigenes ROM, um die physische Dateistruktur (die Anordnung der Datenbytes auf dem Medium) bearbeiten zu können. Diese Struktur wird von OS-Routinen aufbereitet, die die logische Struktur (Datensätze, Felder etc.) festlegen und für alle Speichermedien identisch sind.

Die Routinen der Dateisystem-ROMs werden über Vektoren angesprochen. Wenn Sie ein bestimmtes Dateisystem einsetzen, veranlassen Sie das OS zunächst, sich dem neuen Dateisystem anzupassen. Dabei kann es natürlich passieren, daß einige Abläufe – beispielsweise der Befehl SAVE an ein ROM – in einem bestimmten System nicht möglich sind.

Es gibt eine Vielzahl von Möglichkeiten, ein Dateisystem anzusprechen. Wir haben hier nur die direkten Methoden aufgeführt:

- Wenn Sie das Dateisystem-ROM an den dafür vorgesehenen Steckplatz setzen, wird es bei RESET automatisch aktiviert. Befindet sich mehr als ein Dateisystem-ROM in der Maschine, wird zuerst der Chip angesprochen, der unmittelbar links von BASIC-ROM liegt – vorausgesetzt, das BASIC-ROM wurde an den rechtsaußen liegenden Steckplatz für paged ROMs gesetzt.
- BREAK mit Zusatztaste: Wenn sich das entsprechende ROM in der Maschine befindet,



aktivieren BREAK und die Taste N das Econet-Dateisystem.

● Sternbefehle: Der Befehl *TAPE12 aktiviert das Cassettensystem mit 1200 BAUD, *DISC das Diskettensystem, *TELESOFT das Telesoftsystem und *NET das Econetsystem.

Das Dateisystem wird bei jedem Einschalten der Maschine aktiviert, selbst wenn es nur für das Laden oder Speichern eines Programms benutzt wird. Die beiden BASIC-Befehle SAVE und LOAD verwenden ebenso wie CHAIN, OPENUP, OPENOUT, OPENIN, INPUT, PRINT etc. die Dateiroutine des OS. Auch einige *-Befehle arbeiten mit Dateisystemroutinen, die meisten greifen jedoch auf die speziellen ROMs ihres eigenen Dateisystems zu.

Wir können nicht alle Einsatzmöglichkeiten der Routinen aufführen, die für Dateivorgänge des Acorn B gebraucht werden. Wir werden daher nur auf die wichtigsten eingehen.

Sieben OS-Routinen (OSFILE, OSARGS, OSBGET, OSBPUT, OSFSC, OSFIND und OSGBPB) und einige Aufrufe von OSWORD und OSBYTE bearbeiten Dateivorgänge.

Attribute

Da OSFILE mit vollständigen Dateien arbeitet, bietet es sich an, zuerst auf diese Routine einzugehen. OSFILE wird von Befehlen eingesetzt, die wie SAVE, LOAD, *SAVE etc. große Datenblöcke verarbeiten. Die Dateien enthalten außer ihren Daten auch sogenannte „Attribute", die Auskunft über die Ladeadresse (LOAD ADDRESS), Ausführadresse (EXECUTION ADDRESS) und die Dateilänge geben. Mit OSFILE kann der Programmierer diese Attribute steuern. Wie OSWORD braucht auch OSFILE einen Parameterblock, dessen Adresse mit den Registern X und Y an OSFILE übergeben wird. OSFILE arbeitet mit Vektoren und wird über die Adresse &FFDD aufgerufen. Der Ablauf bei Routine OSFILE hängt davon ab, welcher Wert sich beim Aufruf im A-Register befindet. Die erste der beiden folgenden Tabellen zeigt die Struktur des Parameterblocks von OSFILE und die zweite die möglichen Abläufe.

OSFILE benötigt für die Angabe der Speicherstellen nur eine 16-Bit-Adresse. In diesem Fall sind nur das LSB und das nächsthöhere Byte der Adreßangabe von Bedeutung. In der zweiten Tabelle wird deutlich, daß OSFILE für Cassetten längst nicht soviele Möglichkeiten bietet wie für ein Diskettensystem. Die Dateiattribute hängen von dem eingesetzten Dateisystem ab. Bei einem Diskettensystem ist es sehr praktisch, beispielsweise die Ladeadresse einer Datei durch eine Änderung des entsprechenden Eintrags im Parameterblock versetzen zu können. Dafür braucht nur der entsprechende Wert in A gespeichert und OSFILE aufgerufen werden.

Das folgende Programm speichert sich über einen OSFILE-Aufruf selbst:

Parameterblock für den Aufruf von OSFILE

Offset von der Adresse in den X- und Y- Registern	Beschreibung
0	LSB der Adresse des Dateinamens
1	MSB der Adresse des Dateinamens
2	LSB der Ladeadresse der Datei
3	keine Angabe
4	keine Angabe
5	MSB der Ladeadresse der Datei
6	LSB der Ausführadresse der Datei
7	keine Angabe
8	keine Angabe
9	MSB der Ausführadresse der Datei
10	LSB der Adresse der Daten, die gespeichert werden, oder Bestimmungsadresse für gelesene Daten
11	keine Angabe
12	keine Angabe
13	MSB der Datenadresse
14	LSB der Endadresse Daten + 1 für Schreibvorgänge oder Informationen über Dateiattribute bei Lesevorgängen
15	keine Angabe
16	keine Angabe
17	MSB der Datenadresse

Abläufe von OSFILE

Inhalt von A	Beschreibung der Abläufe	
	Diskette	Cassette
0	Speicherbereich sichern	Speicherbereich sichern
1	In den Katalog schreiben	Datei laden
2	Ladeadresse schreiben	Datei laden
3	Ausführadresse schreiben	Datei laden
4	Dateiattribute schreiben	Datei laden
5	Katalog lesen	Datei laden
6	Datei löschen	Datei laden
255	Datei laden	Datei laden

```
10 DIM block 20
20 DIM name 10
30 BASIC=&8023:REM für BASIC II, &801F
   für BASIC I
40 X%=block MOD 256:REM Zeiger auf den
   Parameterblock
50 Y%=block DIV 256
```

```
60 A%=0:REM Schreibvorgang angeben
70 block!0=name:REM Addresse des
   Dateinamens
80 block!2=PAGE:REM Ladeadresse des
   Programms
90                 :REM wird PAGE
100 block!6=BASIC:REM Ausführadresse
   wird der
110                :REM Einsprungspunkt für
   BASIC
120 block!10=PAGE:REM Erstes
   geschriebenes Byte
130                :REM ist PAGE
140 block!14=(TOP+1):REM Ende der
   Daten +1
150 $name="FRED":REM setzt einen legalen
   Dateinamen ein
160 CALL &FFDD:REM ausführen
170 END
```

Auch das Löschen eines Dateinamens ist einfach – man muß den Namen in den Speicher laden, mit den ersten zwei Einträgen des Parameterblocks darauf zeigen, das A-Register auf 6 setzen und OSFILE aufrufen. Nach einem Aufruf von OSFILE mit A=5 enthalten Byte 2 und 5 des Parameterblocks die Ladeadresse der Datei und Bytes 6 und 9 die Ausführadresse. Wenn für den Namen, auf den die Adresse des Parameterblocks zeigt, kein Eintrag auf der Diskette gefunden wird, dann enthält A bei der Rückkehr von der Routine den Wert 200. Wird eine Datei gefunden, steht A auf 1.

*-Befehle der OS-Dateisystemroutinen

*Befehl	Beschreibung
*CAT oder *	Legt einen Katalog der Dateien an, die im Augenblick auf dem Dateisystem verfügbar sind.
*SAVE	Schreibt einen Byteblock vom Arbeitsspeicher auf das permanente Speichermedium. Beispiel: *SAVE "Dateiname" Anfang Ende Ausführung "Anfang" ist die Hexadezimaladresse des ersten Bytes, das gespeichert werden soll, und "Ende" die Hexadezimaladresse des letzten Byes +1. "Ausführung" ist die Ausführadresse der so entstandenen Bytedatei. „Ausführung" ist ein Parameter, der wahlweise eingesetzt werden kann. Wenn er fehlt, wird die Ausführadresse auf die Startadresse gesetzt. "Ende" kann durch "+Länge" ersetzt werden, das die Anzahl der Bytes angibt, die gespeichert werden sollen.
*LOAD	Lädt eine Bytedatei an die angegebene Adresse im Arbeitsspeicher. Der Befehl hat folgendes Format: *LOAD "Dateiname" Adresse "Adresse" ist die Hexadezimaladresse der Position, an die die Bytes geladen werden sollen. Fehlt der Parameter, dann werden die Bytes an die Adresse geladen, von der aus sie gesichert wurden.
*RUN	Entspricht einer Maschinencodeversion des BASIC-Befehls CHAIN. *RUN "Dateiname" lädt eine Bytedatei an die Adresse, von der aus sie gespeichert wurde und führt sie als Maschinencodeprogramm von der beim SAVE angegebenen Adresse an aus.
*SPOOL und *EXEC	Diese beiden Befehle wurden schon früher erklärt.
*OPT1,x	Dieser Befehl teilt dem OS mit, ob Meldungen dargestellt werden sollen. Bei x=0 werden keine Meldungen angezeigt; x=1 liefert kurze, x=2 längere Meldungen.
*OPT2,x	Dieser Befehl steuert die Abläufe des OS, wenn während eines Dateivorgangs ein Fehler auftritt. Bei x=0 wird der Fehler ignoriert, x=1 teilt dem Anwender mit, es nochmals zu versuchen (der Fehler wurde gefunden), und x=3 veranlaßt bei einem Fehler den Abbruch des Dateivorgangs.
*OPT3,x	Legt Pausen fest, die beim Schreiben auf Cassette eingesetzt werden, x gibt Einheiten von 0,1 Sekunden an.
*OPT4,x	Dieser Befehl wird nur bei Diskettensystemen eingesetzt. Er setzt auf der Diskette die SHIFT-BREAK-Optionen (siehe betreffendes Handbuch).

Byte für Byte

Dieser Artikel beschließt unsere Untersuchung der ROM-Routinen des Dateisystems für den Acorn B. Wir untersuchen OSFIND und verwandte Routinen, mit denen wir Zugriff auf einzelne Bytes haben.

OSFIND wird über die Adresse &FFCE aufgerufen und eröffnet eine Datei, um auf einzelne Bytes zugreifen zu können. Dabei geschieht das gleiche wie bei der Eröffnung einer Datei mit OPENOUT, OPENIN oder OPENUP von BASIC aus. Weiterhin lassen sich damit Dateien auf die gleiche Weise schließen wie mit CLOSE# in BASIC. Das A-Register gibt an, welche Funktion ausgeführt wird, während X und Y auf den Dateinamen im Speicher zeigen.

●**A=0:** Zeigt an, daß die Datei geschlossen werden soll. Das Y-Register enthält dabei die „Nummer", die der Datei bei ihrer Eröffnung zugeteilt wurde. Die Angabe der Dateinummer 0 schließt alle geöffneten Dateien.

●**A=&40:** Entspricht dem BASIC-Befehl OPENIN. Eine Datei wird nur für die Eingabe von Daten(bytes) geöffnet.

●**A=&80:** Entspricht dem BASIC-Befehl OPENOUT. Eine Datei wird nur für die benötigte Ausgabe geöffnet.

●**A=&C0:** Entspricht dem BASIC-Befehl OPENUP. Eine Datei wird für Ein- und Ausgabe eröffnet. Wird für Dateisysteme eingesetzt, die Random-Access-Dateien unterstützen.

A wird auf Null gesetzt

Falls OSFIND eine Datei nicht eröffnen kann (beispielsweise bei einer schreibgeschützten Diskette), wird A bei der Rückkehr von OSFIND auf Null gesetzt. Ist die Eröffnung jedoch erfolgreich, erhält A eine Dateinummer, die für einen späteren Einsatz zur Verfügung steht.

Alle folgenden Aufrufe von ROM-Routinen des Dateisystems haben eine Eigenschaft gemeinsam: Sie verwenden die mit OSFIND erzeugte Dateinummer.

Die OSARGS-Routine wird über &FFDA aufgerufen und läßt sich für eine ganze Reihe von Aufgaben einsetzen. Sie hat eine Eigenheit: Ihr Parameterblock von nur vier Byte Länge ist auf der Zero Page untergebracht. Das X-Register enthält dabei die Adresse des ersten Parameterblockbytes, Y die Dateinummer und A die Nummer der Funktion, die ausgeführt werden soll. Ein spezieller Fall von OSARGS tritt ein, wenn das Y-Register auf Null gesetzt ist. In diesem Fall liefert die Routine Informationen über das Dateisystem statt über eine bestimmte Datei oder veranlaßt, daß alle eröffneten Dateien des Dateisystems mit den Daten aktualisiert werden, die sich noch in den Dateibuffern befinden. Hier zunächst diese beiden Routinen:

●**Y=0:A=0:** Wenn OSARGS mit diesen Parametern aufgerufen wird und das X-Register auf einen freien Bereich der Zero Page zeigt, enthält das Register A nach dem Rücksprung Informationen über das Dateisystem. Dies ist der einzige Aufruf von OSARGS, der sich für Cassettensysteme eignet. Unsere Tabelle am Rand zeigt, wie die Daten, die nach der Rückkehr in A stehen, interpretiert werden.

●**Y=0:A=255:** Bei dem zweiten Aufruf mit Y auf Null enthält A den Wert 255. Dieser Aufruf von OSARGS stellt sicher, daß alle Daten, die sich noch in den Buffern der Maschine befinden, in die entsprechenden Dateien geschrieben werden. Sind keine Dateien eröffnet, kann auch keine Aktualisierung stattfinden.

●**Y=Dateinummer:** Wenn Y die Dateinummer enthält, bestimmt der Inhalt des Registers A, welche Funktion ausgeführt wird (siehe untenstehende Tabelle). Der „sequentielle Zeiger" auf eine offene Datei informiert das Betriebssystem darüber, wohin das nächste Byte geschrieben oder von wo es abgerufen werden soll. Der Zeiger läßt sich daher von der BASIC-Funktion PTR# und auch von diesen Routinen ändern. Die Daten werden im Parameterblock auf der Zero Page gespeichert, wobei das LSB in der Adresse (X+0) untergebracht ist. Daten, die OSARGS aus dem Dateisystem liest, werden entsprechend in diesem Parameterblock zwischengespeichert.

Inhalt des A-Registers	Beschreibung
0	Die aktuelle Position des sequentiellen Pointers der im Y-Register angegebenen Datei lesen
1	Den Parameterblockwert in den sequentiellen Pointer schreiben. Ähnlich dem Befehl PTR#n=Wert
2	Dateilänge in den Parameterblock einsetzen
255	Datei (deren Nummer im Y-Register steht) auf dem Speichermedium aktualisieren

Aus der Beschreibung wird deutlich, warum der Parameterblock von OSARGS sich auf der Zero Page befinden muß: Für die Adresse steht nur das X-Register zur Verfügung.

Mit den Routinen OSBGET und OSBPUT wird

Inhalt des A-Registers	Beschreibung
0	Kein Dateisystem
1	Band mit 1200 Baud
2	Band mit 300 Baud
3	ROM
4	Diskette
5	Netzwerk
6	Telesoft

ein Byte aus einer offenen Datei gelesen oder dorthin geschrieben. Der Aufruf von OSBGET erfolgt über &FFD7. Dabei wird das Y-Register mit der von OSFIND angelegten Dateinummer geladen und über den Aufruf von &FFD7 ein Byte von der aktuellen Position des sequentiellen Pointer gelesen und in A gespeichert.

OSBPUT ist die Umkehrung dieses Vorgangs. Die Routine schreibt ein Byte an die Position der Datei, auf die der sequentielle Pointer zeigt. Das Byte, das geschrieben werden soll, wird dafür in A gespeichert und die Dateinummer in Y. Der Aufruf von &FFD4 führt die Routine aus. Das folgende Modul zeigt, wie OSBPUT ein einzelnes Byte speichert.

```
LDA   #&80 / Dateieröffnung vorbereiten
LDX   #name MOD 256
LDY   #name DIV 256
JSR   OSFIND /Datei öffnen
STA   &71 / Dateinummer speichern
TAY   / und im Y-Register ablegen
LDA   #65 / Das Byte, das geschrieben werden soll
JSR   OSBPUT / schreiben
LDA   #0
LDY   &71 / Dateischließung vorbereiten
JSR   OSFIND / Datei schließen
RTS   / Ende
```

Die OSGBPB-Routine wird über &FFD1 aufgerufen. Sie transferiert Bytegruppen zwischen dem Speicher und offenen Dateien und kann auch Informationen über das eingesetzte Dateisystem liefern. Wir werden hier jedoch nur auf ihre Funktionen bei der Datenübertragung eingehen. Die Routine läßt sich nicht mit Cassettensystemen einsetzen, da sie sich eigentlich nur für den Zugriff auf Random-Access-Dateien vom Maschinencode aus eignet.

Die vorstehende Tabelle stellt den Parameterblock des Aufrufs dar, auf dessen Adresse die Register X und Y zeigen. A gibt an, welche Funktion eingesetzt werden soll.

Bei einem standardmäßigen Acorn B wird die Datenadresse vom Inhalt des Bytes 1 und 4 des Parameterblocks angegeben.

Parameterblock von OSGBPB	
Adresse	**Beschreibung**
0	Dateinummer, wie von OSFIND geliefert
1	LSB der Adr. der Daten (READ/WRITE)
2	Nicht festgelegt
3	Nicht festgelegt
4	MSB der Datenadresse
5	LSB der Anzahl Bytes, für oder aus Datei
6	Nicht festgelegt
7	Nicht festgelegt
8	MSB der Anzahl der zu bewegenden Bytes
9	LSB der Position des sequentiellen Pointers (welche Daten gelesen werden sollen)
10	Nicht festgelegt
11	Nicht festgelegt
12	MSB der Position des seq Pointers

```
10REM disk system only
20DIM  block 20
30DIM data 100
40$data="This is a test program!"
50 block!1=data:REM address of data to be
 written
60block!5=100:REM amount of data to be written
70block!9=0:REM sequential pointer=0
80 PRINT "Opening the file for output"
90Y%=OPENOUT("FILE"):REM open the file
100block?0=Y%          :REM file number
110X%=block MOD 256
120Y%=block DIV 256
130A%=1: REM prepare to write data
140 CALL &FFD1 :REM  do it
150CLOSE £(block?0)
160 PRINT "Close the file"
170 TIME =0: REPEAT: UNTIL TIME=400
180 PRINT "Opening the file for input"
190PRINT "Now read the data in!"
200$data=STRING$(100," "):REM clear data area
210Y%=OPENIN("FILE"):REM open the file up
220block?0=Y%          :REM file number
230block!1=data :REM data address for read
operation
240block!5=100 : REM number of bytes
250block?9=0 : REM sequential pointer to 0
260X%=block MOD 256
270Y%=block DIV 256
280A%=3: REM prepare to read
290CALL &FFD1 :REM do it
300CLOSE£(block?0)
310 PRINT "Close the file again"
320PRINT $data
330END
```

Es stehen vier verschiedene Übertragungsarten zur Verfügung: zwei Lese- und zwei Schreibvorgänge. Je ein Lese- und Schreibvorgang spricht die Daten der Position an, die der sequentielle Pointer des Parameterblocks angibt. Die anderen beiden Routinen ignorieren diesen Eintrag und greifen statt dessen auf die Position zu, auf die der Pointer gerade zeigt. A=1 schreibt daher Bytes an den im Block angegebenen Pointer, während A=2 den Pointer ignoriert und Bytes an die aktuelle Pointerposition schreibt. A=3 liest Daten von der im Block angegebenen Position, während A=4 Bytes liest, aber die Pointeradresse des Blocks ignoriert. Das folgende Programm setzt OSGBPB ein, um eine einfache Datei zu schreiben.

Der Status des Übertragsflags zeigt nach dem Ablauf von &FFD1, ob die Übertragung erfolgreich war oder nicht. Wenn C Null enthält, verlief sie reibungslos, steht C auf Eins, fand keine Übertragung statt.

Die letzte Betriebssystemroutine für Dateivorgänge ist OSFSC. Sie hat für eine Programmierung jedoch nur wenig Nutzen. Ihr Aufruf unterscheidet sich von den zuvor erläuterten, da er nicht über eine direkte Adresse stattfindet, sondern über die Adresse seines Vektors. In einem Maschinencodeprogramm sieht das so aus:

```
JMP (&21E)
```

Weitere OS-Routinen führen Spezialfunktionen für bestimmte Dateisysteme aus.

Zwei Versionen

In dieser und der nächsten Folge untersuchen wir, wie BASIC mit dem Betriebssystem des Acorn B kommuniziert und nehmen die Fehlerbearbeitungsroutinen des Rechners aufs Korn.

Das BASIC-ROM des Acorn B belegt die Speicheradressen zwischen &8000 und &BFFF. Dieser Bereich wird auch als „Paged ROM" bezeichnet. Dabei können sich mehrere verschiedene ROM-Chips in der Maschine befinden, die mit dem gleichen Speicherbereich arbeiten, von denen jedoch nur jeweils einer aktiv sein kann. Beispiele sind der DFS-Chip, der Econet-Chip, Programm-Chips wie Wordwise, View oder Disk Doctor und Sprachen wie FORTH oder BCPL. Die Technik der Paged ROMs spart viel Speicherplatz.

Der Wert des Paging-Registers bestimmt, welches Paged ROM aktiv ist. Dieses Register ist Teil der Hardware, die vom OS gesteuert wird. Es kann bis zu 16 Paged ROMs aufrufen. Wenn das OS einen Befehl nicht identifizieren kann, spricht es zunächst die Paged ROMs an. Bei einem unbekannten *-Befehl werden daher erst die Paged ROMs untersucht, bevor die Fehlermeldung „BAD COMMAND" erscheint. Beispielsweise wird *INFO vom OS-ROM nicht erkannt und an die Paged ROMs weitergereicht. Ist das DFS-ROM installiert, wird der Befehl akzeptiert und ausgeführt.

Informatives Copyright

Für den Acorn B gibt es zwei BASIC-Versionen: BASIC I und BASIC II. Um die Version Ihrer Maschine feststellen zu können, betätigen Sie CTRL-Break, geben REPORT ein und drücken Return. Enthält die Copyright-Meldung das Datum 1982, arbeiten Sie mit BASIC II, bei 1981 mit der älteren Version BASIC I. Im neueren Dialekt wurden einige Fehler des BASIC I korrigiert. Es hat außerdem zusätzliche Fähigkeiten zur Ausführung bereit.

Neu sind unter anderem der Dateibefehl OPENUP und die modifizierte Funktion von OPENIN. In BASIC II eröffnet OPENIN eine Datei nur für die Eingabe, während in BASIC I damit die Ein- und Ausgabe möglich war – eine Fähigkeit, die sich für Random-Access-Dateien eignet. BASIC II hat für diesen Zweck den Befehl OPENUP, der Dateien für die Ein- und Ausgabe eröffnet und dem OPENIN des BASIC I entspricht. Diese Änderung verursacht Probleme bei der Übertragung von Programmen zwischen Maschinen mit unterschiedlichen BASIC-Versionen, da der Token, der OPENIN in BASIC II darstellt, von BASIC I nicht erkannt wird. Der BASIC-II-Befehl:

```
Y%=OPENIN("FRED")
```

erscheint in BASIC I als

```
Y%=("FRED")
```

und ruft dort eine Fehlermeldung hervor.

Einige neue Eigenschaften von BASIC II sind besonders für den Maschinencodeprogrammierer interessant. Eine Verbesserung sollte besonders erwähnt werden. Um unter BASIC I Daten in Maschinencodeprogramme einsetzen zu können, mußte erst der Assembler verlassen und die Operationen ?, ! und $ eingesetzt werden. Hier ein Beispiel:

```
1000 LDA data
1010 .data
1020 ]
1030 ?P%=00
1040 P%=P%+1
1050 OPT pass
```

Die meisten Assembler bieten statt dieser umständlichen Methode „Assembleranweisungen", die den Assembler veranlassen, bestimmte Aufgaben auszuführen. In unserem Programmbeispiel ist OPT eine derartige Anweisung. BASIC II bietet vier weitere Befehle dieser Art: EQUB reserviert ein einzelnes Speicherbyte, EQUD zwei Bytes, EQUW vier Bytes, und mit EQUS kann ein String im Speicher angelegt werden. Das BASIC-I-Programm sieht daher in BASIC II so aus:

```
1000 LDA Daten
1010 .Daten EQUB 00
1020 . . . weitere Programmteile
```

Dabei wird das Register A mit dem Byte der Adresse „Daten" geladen (hier eine Null). Ein weiteres Beispiel:

```
1000 .Meldung EQUS "Hallo!"
```

Hier werden die Bytes, die die Meldung „Hallo!" darstellen, im Speicher von der Adresse mit dem Label .Meldung an gespeichert. Bei all diesen Assembleranweisungen wird P% automatisch hochgezählt.

Auch der BASIC-II-Befehl OSCLI ist sehr praktisch. Wir hatten bereits erwähnt, daß *-Befehle per OSCLI (bei &FFF7) an das Betriebssystem übergeben werden. In BASIC I mußten dafür erst die Variablen X% und Y% (die Register X und Y) mit der Adresse des Befehlsstrings geladen werden. Der neue OSCLI-

Befehl ist weit einfacher. Vergleichen Sie einmal die beiden Versionen:

BASIC I	BASIC II
S&A00="*TAPE" X%=&A00 MOD 256 Y%=&A00 DIV 256 CALL &FFF7	OSCLI("*TAPE")

Außer den zwei erwähnten BASIC-Versionen gibt es noch ein BASIC II für den amerikanischen Markt, das die andersartige Bildschirmdarstellung berücksichtigt. Zudem besitzt der zweite 6502-Prozessor eine eigene BASIC-Version namens „HI BASIC", die eine Mischung der BASIC II US-Version und BASIC II ist.

Speichertopographie

Zwischen &400 und &7FF ist der Speicher für die aktuelle Sprache reserviert („Language Workspace"). Auch auf der Zero Page stehen die Adressen zwischen &00 und &6F der aktuellen Sprache zur Verfügung, wenn auch der Platz von &50 bis &6F nicht oft eingesetzt wird. Auf Speicherseite 4 liegen die BASIC-Variablen: zwischen &400 und &46B die Werte für die eingebauten Ganzzahlvariablen wie A% oder X%. @% wird in den vier Bytes von &400 bis &403 gespeichert (mit dem niederwertigen Byte in &400), A% von &404 bis &407 – und so weiter bis Z%.

Zwischen &480 und &4F9 befindet sich der Index des BASIC-Interpreters. Er gibt die Bereiche an, in denen Arrays und die anderen Fließkomma- oder Stringvariablen gespeichert sind. So zeigen beispielsweise die Bytes in &482 und &483 auf einen Speicherbereich zwischen TOP und HIMEM, in dem sich die Daten aller zur Zeit eingesetzten Variablen befinden, deren Namen mit A beginnen. Die in &4F6 und &4F7 gespeicherte Adresse zeigt auf den Speicher der BASIC-Prozeduren und die Adresse in &4F8 und &4F9 auf den Bereich für Funktionen. Auf diese Weise sind die Übersichtlichkeit und der geregelte Zugriff jederzeit sicher gewährleistet.

Schleifeninformationen

Das RAM zwischen &500 und &5FF dient dem BASIC-Interpreter als Stack (darf nicht mit dem Maschinenstack des 6502 verwechselt werden, der auf Speicherseite 1 liegt). Die Seite 5 verwendet BASIC für die RETURN-Daten der GOSUBs. Dort liegen auch alle Informationen, die für die Ausführung von FOR . . .NEXT- und REPEAT . . .UNTIL-Schleifen nötig sind.

Ihre Ausführung verlangt vom BASIC-Interpreter recht komplexe „Denkleistungen". Sobald etwa FOR im Programm auftritt, muß sich der Rechner merken, wohin er beim zu erwartenden NEXT zurückspringen muß.

Die Seite 6 setzt BASIC entweder für Stringvariablen ein oder für die Ausführung von Maschinencoderoutinen, die mit der erweiterten Version von CALL aufgerufen werden. Seite 7 dient als Eingabebuffer für BASIC-Befehle oder Programmzeilen. Die Eingaben werden dort bis zu ihrer Weiterverarbeitung gespeichert. Direkte Befehle werden interpretiert und ausgeführt und Programmzeilen an der entsprechenden Position in das Programm eingesetzt. Wenn dieser Speicherbereich nicht für BASIC benötigt wird, steht er für andere Zwecke zur Verfügung.

Sicher haben Sie schon gesehen, wie Ihr Computer einen fehlerhaften Befehl mit einer Fehlermeldung quittierte. Auf dem Acorn B können Sie eigene Fehleranweisungen mit Fehlernummer und Meldung anlegen. Diese Möglichkeit ist besonders für Maschinencodeprogramme praktisch, die zusammen mit BASIC eingesetzt werden. Der Inhalt des Vektors BRKV (bei &202 und &203) zeigt auf die Routine, die die von BASIC-Programmen erzeugten Fehler bearbeitet.

Kommentierbare Fehler

Sobald BASIC einen Fehler entdeckt – beispielsweise bei einer Teilung durch Null –, wird der Maschinencodebefehl BRK ausgeführt. Dadurch spricht der 6502 eine Routine an, deren Adresse in BRKV gespeichert ist. Auf den BRK-Befehl müssen eine Reihe Bytes folgen, die folgendes Format haben:

BRK
Fehlernummer (ein Byte)
Fehlermeldung (eine Bytefolge)
Ende der Meldung (ein Null-Byte)

Fehler Nummer 4 (der die Meldung „Mistake" ausgibt) wird mit folgender Routine bearbeitet:

Byte	Beschreibung
BRK	Befehl
4	Fehlernummer
M	
i	
s	ASCII-Code der Fehlermeldung
t	
a	
k	
e	
00	Ende der Meldung

Das BASIC-Fehlermodul setzt nun die Systemvariable ERL auf die Nummer der Zeile, in der der Fehler auftrat, und ERR auf die Fehlernummer – in diesem Fall 4. Das Fehlermodul stellt sicher, daß bei der nächsten Bewertung von REPORT die Fehlermeldung „Mistake" ausgegeben wird.

Der Befehl BRK läßt sich aber auch in selbsterstellten Maschinencodeprogrammen einsetzen, und somit können eigene Fehlermeldungen definiert werden. Der BASIC-Befehl ON ERROR GOTO behandelt diese neu generierten Meldungen wie die normalen, vom BASIC-Interpreter gemeldeten Fehler. So verursacht der folgende Maschinencode die Fehlermeldung „Zahl zu groß":

```
1010 BRK
1020 EQUB 254
1030 EQUS "Zahl zu groß"
1040 EQUB 00
```

Nach der Ausführung erhält ERR die Zahl 254 und REPORT die Meldung "Zahl zu groß".

Da der Inhalt dieses Vektors verändert werden kann, läßt sich die gesamte Fehlerbehandlung des Acorn B umstellen. Es besteht sogar die Möglichkeit, neue Befehle an das BASIC des Acorn B anzufügen – die Programmierung dafür ist jedoch recht kompliziert.

In der nächsten Folge sehen wir uns an, wie BASIC II mit dem Betriebssystem des Acorn B arbeitet und welche Vorteile es bei seiner Kooperation mit dem OS gegenüber seinem Vorgänger tatsächlich bietet.

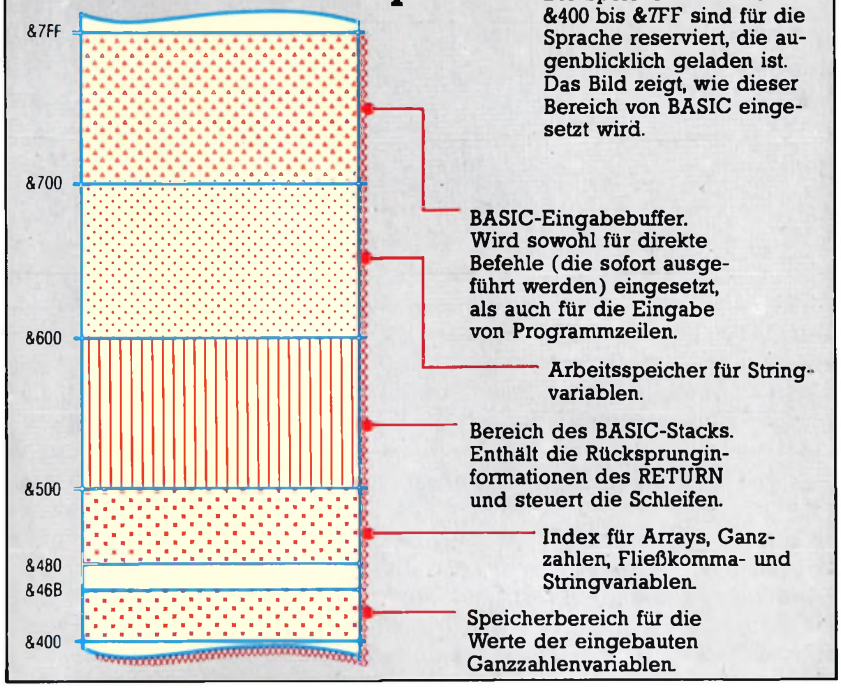
Routineuntersuchung

Die Liste zeigt die ROM-Adressen der BASIC-Routinen des Acorn B, die wir in dieser Serie behandelt hatten. Wenn Sie sich die Routinen genauer ansehen, erhalten Sie einen guten Einblick in die internen Arbeitsabläufe des BASIC. Da nicht alle hier aufgeführten Routinen mit dem Befehl RTS enden, empfehlen wir, sie nacheinander von einem eigenen Programm aus aufzurufen. Die Analyse dieser Techniken kann Ihnen wertvolle Hinweise für Ihre eigenen Programme liefern.

ROM-Einstiegsadresse	**&8000**
BGET Anweisung	&BF6F
BPUT Anweisung	&BF58
CHAIN Anweisung	&BF2A
CLOSE Anweisung	&BF99
EOF Funktion	&ACB8
INKEY Funktion	&ACAD
INKEY$ Funktion	&B026
INPUT Anweisung	&BA44
INPUT# Anweisung	&B9CF
LOAD Befehl	&BF24
OPENIN Funktion	&BF78
OPENOUT Funktion	&BF7C
OPENUP Funktion	&BF80
PRINT# Anweisung	&8D2B
SAVE Befehl	&BEF3
TIME Anweisung	&92C9
TIME Funktion	&AEB4
VDU Anweisung	&942F

Diese Information wurde mit freundlicher Genehmigung von Acorn Computers abgedruckt
(C) Acorn Computers Ltd (1982)

Arbeitsbereich der Sprache



Unterschiede zwischen BASIC I und BASIC II

Befehl oder Fehler	BASIC I	BASIC II
REPORT	Liefert die Meldung 1981	Liefert die Meldung 1982
OPENUP	Nicht vorhanden	Eröffnet eine Datei für Lese- und Schreibvorgänge (Wie OPENIN in BASIC I)
OPENIN	Eröffnet eine Datei zum Lesen und Schreiben	Eröffnet eine Datei nur zum Lesen
OPT n	Es gibt nur OPT 1 bis 3	Es gibt OPT 1 bis 7. Die vier neuen Befehle (OPT 4 bis 7) haben die gleichen Funktionen wie die des BASIC I, werden jedoch auf der Adresse abgelegt, die in der Variablen 0% gespeichert ist
0%	Keine spezielle Bedeutung	Hat mit OPT (4 bis 7) die dort beschriebene Funktion
EQUB EQUD EQUS EQUW	Nicht vorhanden	Erlaubt dem Programmierer, Daten in Maschinencodeprogramme einzusetzen, ohne den Assembler verlassen zu müssen
INSTR (a$,b$)	Ein schwerer Fehler. Wenn a$ kürzer war als b$, bestand die Möglichkeit eines Programmabsturzes	Der Fehler wurde korrigiert
reelle Zahlen	Genauigkeit: neun Stellen	Genauigkeit: zehn Stellen
OSCLI	Nicht vorhanden	Eine einfache Möglichkeit, per BASIC String-Variablen-Befehle an das OS zu übergeben

Kooperation

In dieser Folge untersuchen wir, wie BASIC mit dem Betriebssystem des Acorn B zusammenarbeitet, und stellen USR und CALL vor.

Es wurde bereits erwähnt, daß viele OS-Routinen auch vom BASIC eingesetzt werden. So ruft der BASIC-Befehl SOUND für die Tonerzeugung OSWORD mit A=7 auf, während ENVELOPE OSWORD mit A=8 auf die gleiche Weise verwendet. Die BASIC-Abläufe lassen sich bis zu einem gewissen Grad beeinflussen, wenn man das Betriebssystem ändert – das heißt durch Vektorveränderungen und durch eigenen Code, der anstelle der bereits besprochenen OS-Routinen abläuft.

OSWRCH, OSASCII und OSNEWL werden von einer Reihe Bildschirmsteuerroutinen eingesetzt. Oft ist jedoch Maschinencode, der Grafik mit OS-Routinen erzeugt, nicht wesentlich schneller als die entsprechenden BASIC-Befehle. Daran läßt sich messen, auf welche Weise der BASIC-Interpreter OS-Routinen bei Befehlen wie MOVE, DRAW und PLOT einsetzt.

Sehen wir uns die beiden BASIC-Befehle einmal an, mit denen Sie von BASIC aus OS-Routinen aufrufen können: USR und CALL.

USR führt eine Maschinencoderoutine aus und kehrt dann mit einem Wert ins BASIC zurück, der Aufschluß über die Register A, X und Y, den Status des Übertragsflag und des Prozessor Status Registers (PRS) gibt. Der Befehl hat folgendes Format:

```
Ergebnis%=USR(Adresse%)
```

Adresse% ist die Speicheradresse der auszuführenden Routine. Bevor der Befehl eingesetzt wird, kann man Werte an A%, X% und Y% übergeben, die von der USR-Routine in die Register A, X und Y gestellt werden. Mit dem niederwertigsten Bit von C% läßt sich außerdem das Übertragsflag beeinflussen – ein gerader Wert von C% setzt es auf Null.

Das folgende Programmbeispiel zeigt den Einsatz von USR. Dabei wird eine Routine von OSBYTE aufgerufen, die im X-Register die X-Koordinate des Cursors ablegt und im Y-Register die Y-Koordinate. Der Wert von Ergebnis% zeigt diese Registerinhalte in einer codierten Form, auf die wir später noch genauer bei unserer ausführlichen Betrachtung des Betriebssystems des Acorn B eingehen.

```
10 A%=134
20 X%=0
30 Y%=0
40 LET Ergebnis%=USR(&FFF4)
50 PRINT Ergebnis%: REM zeigt eine Hexzahl
```

Die ermittelten Werte lassen sich leichter decodieren, wenn das Ergebnis im Hexadezimalformat angezeigt wird.

USR in Aktion

Mit USR läßt sich von BASIC aus ein Maschinencodeprogramm aufrufen, das einen Wert an BASIC zurückgibt. Da USR eine Funktion ist (im Gegensatz zu dem „Befehl" CALL), muß der zurückgegebene Ergebniswert entweder einer Variablen zugeordnet werden

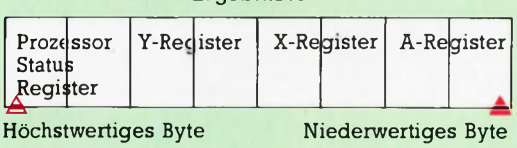
Ergebnis=USR(&FFF4)

oder Teil eines PRINT-Befehls sein:

PRINT USR(&FFF4)

Der von USR an BASIC gelieferte Ergebniswert ist eine Ganzzahl mit der Länge von vier Bytes, die den Inhalt der Register P, X, Y und A folgendermaßen speichert:

Ergebnis%=



Höchstwertiges Byte — Niederwertiges Byte

Sie erhalten den Wert eines bestimmten Registers, indem Sie mit AND Teile des Ergebnisses maskieren.

Ergebnis%AND&000000FF

maskiert alle Werte außer Register A.

Maschinencodeprogramme lassen sich auch mit CALL aufrufen. Wir haben diesen Befehl schon mehrere Male verwandt – speziell für den Aufruf von OSWORD und anderen OS-Routinen. Die gewünschte Routine wird über ihre Adresse aufgerufen. Hier ein Beispiel:

```
CALL &FFF1
```

Das BASIC des Acorn B enthält eine erweiterte Version dieses Befehls, die dem Maschinencodeprogramm Werte von BASIC-Variablen übergeben kann. Dabei stellt sich die Frage, wie dem Maschinencodeprogramm mitgeteilt werden soll, an welcher Stelle des Speichers sich die Variablen befinden. Der BASIC-Interpreter löst das Problem, indem er bei einem CALL mit Parametern wie:

```
CALL Adresse%,A%,C%
```

alle Informationen über A% und C% in einem Bereich der Speicherseite 6 ablegt, der „Parameterblock" genannt wird. Die Tabelle zeigt den Aufbau des Blocks:

Stelle	Inhaltsbeschreibung
&0600	Anzahl der Parameter
&0601	Adresse des ersten Parameters
&0602	
&0603	Typ des ersten Parameters
&0604	Adresse des zweiten Parameters
&0605	
&0606	Typ des zweiten Parameters

Selbstverständlich kann der Block beliebig viele Einträge für Adressen und Typen enthalten. Der Adreßparameter ist einfach beschrieben: An dieser Speicherstelle befindet sich das erste Byte des Parameters. Der Typparameter zeigt an, welche Art Variable der Parameter ist. Hier alle verfügbaren Typen:

Type	Beschreibung	Beispiel
0	8-Bit-Byte	CALL Adresse, ?&70
4	Ganzzahlvariable	CALL Adresse, F%
8	Reelle-Variable	CALL Adresse, G
128	String bei Adresse	CALL Adresse, $(&A00)
129	String-Variable	CALL Adresse, F$

Der letzte Eintrag – Typ 129 – fällt aus dem Rahmen heraus. Hier zeigt der Adreßeintrag des Parameterblocks nicht auf die Position des Strings im Speicher, sondern auf einen Bereich, der weitere Informationen über den String liefert – den „String Information Block". Er hat folgendes Format:

Stelle	Inhaltsbeschreibung
Im Parameterblock gespeicherte Adresse	Adresse des String
	zugeteilte Bytes
	aktuelle Stringlänge

Die ersten zwei Bytes des Information Blocks zeigen auf das erste Zeichen des Strings, und der Eintrag für „zugeteilte Bytes" gibt die maximale Länge an, die dem String zugeteilt wurde. Byte 4 enthält seine aktuelle Länge.

Mit CALL kann vom BASIC aus ein Maschinencodemodul ausgeführt werden, das im Gegensatz zu USR allerdings keinen Wert zurückgibt. Sie können jedoch eine Subroutine schreiben, die die Ergebniswerte des Maschinencodeprogramms in BASIC-Variablen speichert, und die sich nach dem Rücksprung weiterverarbeiten lassen.

Das folgende Programm zeigt nicht nur die Parameterübergabe eines CALL-Befehls, sondern stellt die Ergebniswerte auch noch in die Ganzzahlenvariable A%. Die Routine demonstriert weiterhin, wie vom Maschinencodeprogramm erzeugte Fehler die Fehlerbehandlungsroutinen von BASIC einsetzen. Das Programm wurde in BASIC II geschrieben.

```
 10 DIM C 400
 20 zero=&70
 30 FOR I%=0 TO 2 STEP 2
 40 P%=C
 50 [OPT 1%
 60 .code LDA&600
 62 CMP#1
 64 BNE error
 70 LDA&603
 80 CMP#4
 90 BNE error2
100 LDA&601:STA&70
110 LDA&602:STA&71
120 LDY#0
130 LDA&02:STA(zero),Y:INY
140 LDA&03:STA(zero),Y
160 RTS
170 .error BRK
180 EQUB 254
190 EQUS "Parameterzahl nicht korrekt"
200 EQUB 00
210 .error2 BRK
220 EQUB 253
230 EQUS "Falscher Typ"
240 EQUB 00
250 ]:NEXT
260 A%=0:CALL code,A%
270 PRINT "Ende der BASIC-Variablen ist";A%
```

Das Programm gibt die Endadresse der BASIC-Variablen in der Ganzzahlenvariablen zurück, die auch als Parameter eingesetzt wurde (und zeigt damit, wie CALL Werte an BASIC weitergibt). Die Zeilen 60 bis 64 prüfen, ob nur ein Parameter übergeben wurde. Ist diese Bedingung nicht erfüllt, erscheint die entsprechende Fehlermeldung. Zeile 70 bis 90 testen, ob der als Parameter übergebene Wert eine Ganzzahl ist. Falls nicht, wird auch hier eine Fehlermeldung angezeigt. Die Zeilen 100 bis 160 übertragen die Daten der Adressen 2 und 3 in die niederwertigsten Bytes der als Parameter übergebenen Ganzzahlvariablen, während 170 bis 240 auf weitere Fehler eingehen.

Zeile 260 schließlich enthält den Befehl CALL. A% wird zuvor auf Null gesetzt, da das Maschinencodeprogramm die höherwertigen Bytes der Ganzzahlvariablen nicht anspricht. Nach Ablauf der Routine gibt HIMEM-A% die Zahl der Bytes an, die bei Abzug des BASIC-Programms und der Variablen noch frei sind und für Programme zur Verfügung stehen.

Wir unterbrechen dieses Programm...

Interrupts geben Betriebssystemen die Möglichkeit, Prozessorkapazitäten des Computers optimal einzusetzen. In dieser Folge über das OS des Acorn B untersuchen wir die Funktionsweise von Interrupts und Ereignissen.

Ein BASIC-Programm, das auf dem Acorn B abläuft, vermittelt leicht den Eindruck, daß die gesamte Maschine sich der Programmausführung widmet. Das ist jedoch nicht der Fall, das Betriebssystem führt auch ständig andere Funktionen aus, etwa die Abfrage der Tastatur. Diese Aufteilung der Computerkapazität wird durch Interrupts (Unterbrechungen) möglich. Interrupts sind im wesentlichen Signale, die die CPU veranlassen, ihre augenblicklichen Aktivitäten anzuhalten und andere Aufgaben auszuführen. Sobald die CPU diese zweite Aufgabe beendet hat, kehrt sie zum ursprünglichen Ablauf zurück und setzt ihn fort. Im Acorn B steuern Interrupts eine ganze Reihe von Aktivitäten: Tastaturabfragen, die Darstellung der Farben, den Ablauf des Befehls ENVELOPE, den Konvertiervorgang des A/D-Wandlers etc. Sehen wir uns diesen Mechanismus einmal an.

Die Interrupts des Acorn B können von mehreren Hardwarekomponenten erzeugt werden, darunter der serielle Schnittstellenchip, die VIAs (Versatile Interface Adaptors – Schnittstellenadaptoren) für Anwender und System sowie die Schnittstellenhardware für Econet und Diskettenlaufwerk. Diese Bauelemente geben zwei verschiedene Interrupttypen an den 6502-Prozessor weiter. Beide unterbrechen den augenblicklichen Ablauf, doch kann ein Interrupttyp von der CPU ignoriert werden. Unterbrechungen dieser Art werden „maskierbare Interruptanforderungen" („Maskable Interrupt Requests" – IRQs) genannt. Der andere Interrupttyp heißt „nicht-maskierbarer Interrupt" („Non-Maskable Interrupt") oder kurz NMI. Er hat eine hohe Priorität und kann von der CPU nicht ignoriert werden.

Die NMIs des Acorn B sind für Systemkomponenten reserviert, die die unmittelbare Aufmerksamkeit der CPU verlangen, wogegen Geräte, die IRQs ausgeben, ruhig etwas warten können. Auf dem Acorn B lösen nur die Econet-Hardware und das Diskettensystem NMIs aus. Da das OS bei Erhalt eines NMI eine Maschinencoderoutine auf Seite &0D anspricht, sollten Sie nach Installation einer Diskettenschnittstelle nichts mehr auf diese Seite schreiben. Wir werden nicht näher auf NMIs eingehen, da der Acorn B sie nur hierfür verwendet.

Wenden wir uns nun den IRQs zu. Auf dem 6502 werden maskierbare Interrupts mit dem Maschinencodebefehl SEI abgeschaltet und mit CLI wieder aktiviert. Wenn der 6502 ein Interrupt erhält, fragt er zunächst alle Interruptquellen ab, ob die Unterbrechung dort ausgelöst wurde. Nach der Identifizierung der Quelle arbeitet der Prozessor die dafür vorgesehene Interruptroutine ab und informiert die Quelle, daß sie die Interruptanforderung nun „vergessen" kann („Clearing the Interrupt"). Maschinencoderoutinen zur Bearbeitung von Interrupts heißen „Interrupt Service Routinen" oder kurz ISR.

ISR sichert die Werte

Da das unterbrochene Programm nach Behandlung des Interrupts mit den gleichen Registerinhalten weiterarbeiten soll, sichert die ISR als erstes diese Werte und schiebt sie auf den Stapel. Nach Behandlung des Interrupts werden sie wieder in ihre Register zurückgeladen. Inzwischen führt die ISR die entsprechende Interruptroutine aus und löscht vor dem Rücksprung in das unterbrochene Programm noch die Interruptbedingung. Der Löschvorgang ist notwendig, da der 6502 sonst unmittelbar nach dem Rücksprung wieder den gleichen IRQ erhalten würde: Ohne Löschung entsteht eine Endlosschleife.

Sehen wir uns diesen allgemeinen Ablauf einmal im konkreten Fall an. Der Acorn setzt Interrupts für viele Vorgänge ein. Aus diesem Grund dürfen Interrupts auch nicht allzu lange mit SEI abgeschaltet werden. Ein paar Millisekunden bringen noch keine Probleme, doch bei längeren Zeiten arbeitet das OS nicht mehr korrekt. Das folgende Programm zeigt, was geschieht, wenn Interrupts für längere Zeit abgeschaltet werden:

```
10 MODE 2
20 DIM C(100):REM Platz für Maschinen-
   code reservieren
30 FOR I%=0 TO 2 STEP 2
40 P%=C
50 [OPT I%
60 .code SEI
70 RTS
```

```
 80 ]:NEXT
 90 COLOUR 14
100 PRINT "Hallo!"
110 ENVELOPE 1,1,4,-4,3,10,20,20,127,0,
    0,-5,126,126
120 SOUND 1,1,160,200
130 TIME=0
140 REPEAT
150 PRINT TIME,I%
160 I%=I%+1
170 IF TIME>100 THEN CALL code
180 UNTIL FALSE:REM Endlosschleife
```

Beim Ablauf dieses Programms werden Sie bemerken, daß sich schon kurz nach Aufruf der Maschinenroutine „code" die Variable TIME nicht mehr verändert, die Farben aufhören zu blinken und die Bearbeitung von ENVELOPE nicht weitergeht. Die Routine gibt aber immer noch das richtige I% aus und zeigt damit, daß die Maschine nicht abgestürzt ist. Nach Beendigung des Programms sollten Sie aber in jedem Fall CTRL BREAK drücken.

Sobald der 6502 ein IRQ erhält, übergibt er die Steuerung an eine ISR, deren Adresse sich im Vektor IRQV1 befindet. IRQV1 liegt bei &204 und &205. Die über diesen Vektor angesprochene Routine prüft zunächst, ob Hardwareinterrupts vorliegen, die das OS routinemäßig benötigt. Die ISR dieses Vektors bearbeitet die Interruptanforderungen dreier Hardwarekomponenten in dieser Reihenfolge: die des seriellen Schnittstellenchips, der System-VIA und einem Teil der Anwender-VIA.

Der serielle Schnittstellenchip ist ein Teil der Hardware und für die Schnittstelle RS423 und das Cassetteninterface zuständig. Interrupts dieser Quellen betreffen natürlich auch diese Systemteile. Der Chip hat bei diesem Vektor höchste Priorität.

Die Interrupts der System-VIA stehen an zweiter Stelle der „Hackordnung". Die außerordentlich flexible VIA verbindet die CPU mit einer Reihe anderer Systemteile und bietet außerdem Möglichkeiten der Zeitbestimmung. Jeder Tastendruck erzeugt ein Interrupt der System-VIA und informiert damit das OS, daß eine Eingabe bearbeitet werden muß. Weiterhin veranlaßt der Chip eine Unterbrechung, wenn ein voller Bildschirminhalt zum Monitor gesandt werden soll. Dieser Interrupt heißt „Vertical Sync Interrupt" und wird jede Fünfzigstelsekunde ausgelöst.

Auch der A/D-Wandler steht mit der System-VIA in Verbindung. Nach jeder Signalumwandlung löst er ein Interrupt aus und veranlaßt damit das OS, die Werte zu aktualisieren, die für die Rückgabe einer Zahl an die Funktion ADVAL nötig sind. Ein weiterer wichtiger Interrupt der System-VIAs findet jede Hundertstelsekunde statt. Er inkrementiert TIME, aktualisiert den „Interval Timer" (der für die Erzeugung von „Ereignissen" eingesetzt wird), dekrementiert den INKEY-Zeitzähler (der die Verzögerung eines INKEY-Befehls mißt) und bearbeitet einen Teil von ENVELOPE. Der Interrupt veranlaßt das OS weiterhin, alle inzwischen betätig-

Vergleichen Sie einmal die CPU des Acorn B mit dem Kellner eines gutbesuchten Restaurants. Wie der Kellner zunächst die wichtigen Gäste (die Priorität vor anderen verlangen) bedient, dann Tische abräumt und gelegentlich auftretende Schwierigkeiten beseitigt, so muß auch der 6502 seine Aufmerksamkeit auf die „Bedienung" der Peripherie, die Steuerung von Datenübertragung und die Programmausführung aufteilen.

Ein geschickter Kellner richtet seine Aufmerksamkeit auf jede dieser Aktivitäten, vergißt dabei aber nicht, seine hungrigen Gäste mit Speisen zu versorgen. Der 6502 arbeitet auf ähnliche Weise. Er antwortet auf Unterbrechungen („Interrupts") oder Ereignisse („Events"), die entweder von der Hardware oder per Software ausgelöst werden. Nach der Behandlung eines Interrupts setzt der Prozessor seine unterbrochene Arbeit fort.

ten Tasten zu bearbeiten. Andere Interrupts der System-VIA betreffen das Sprachsynthesesystem und (falls vorhanden) den Lichtgriffel. Zwar gibt es OSBYTE-Routinen, mit denen Sie die Reaktion des OS auf Interrupts dieser beiden Quellen ändern können, doch werden Sie gleich sehen, welche schwerwiegende Folgen Veränderungen von IRQV1 haben können.

Die Interrupts mit der niedrigsten Priorität kommen von der Anwender-VIA, also der Systemkomponente, die die Kommunikation mit dem User Port und dem Druckerinterface selbständig steuert.

Jedes Interrupt, das von diesem Vektor nicht bearbeitet werden kann, wird an eine ISR von Vektor IRQV2 weitergegeben. Die meisten Interrupts, die diesen Vektor ansprechen, kommen von der Anwender-VIA, die sich mit einer Reihe von Interrupts an die CPU programmieren läßt. Diese Unterbrechungen würden jedoch von der CPU im Normalfall nicht erkannt und ausgeführt werden.

„Intercepting"

IRQV2 liegt bei den Adressen &206 und &207 und läßt sich ändern, um auf die ISR des Anwenders zu zeigen. Obwohl auch IRQV1 geändert werden kann, sollten Sie für Ihre ISR nur IRQV2 einsetzen. Die Umstellung eines Vektors auf eine andere Routine heißt „Intercepting" (abfangen) eines Vektors. Um den IRQV2-Vektor abfangen zu können, müssen beim Einsprung in eine selbsterstellte Routine folgende Bedingungen erfüllt sein:

- Die CPU muß den Inhalt des A-Registers bereits in &FC gespeichert und das Prozessor Status Register (PSR) auf den Stapel geschoben haben.
- Die Register X und Y sollen in dem Zustand sein, in dem sie sich vor Auftreten des Interrupts befanden.

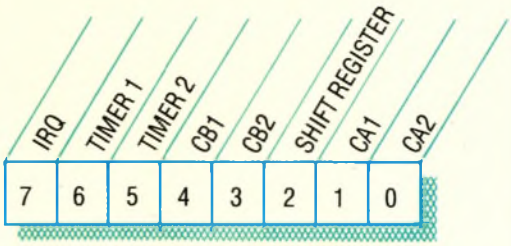
Punkt zwei besagt, daß Sie die X- und Y-Register beim Eintritt in Ihre ISR mit folgenden Befehlen sichern müssen:

```
TXA:PHA
TYA:PHA
```

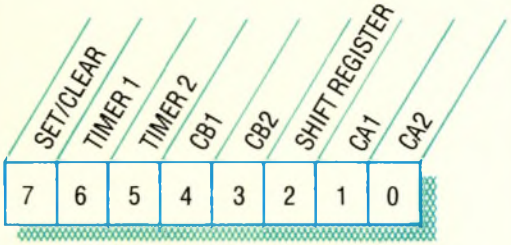
Danach sollte Ihre Routine die Interruptbedingung löschen. Dieser Vorgang hängt jedoch von der Quelle der Unterbrechung ab. Bei der Anwender-VIA müssen dafür gewöhnlich bestimmte Register der VIA gelesen oder geschrieben werden. Denken Sie auch daran, vor der Rückkehr in das unterbrochene Programm die Register wieder in ihren ursprünglichen Zustand zu versetzen und die ISR mit dem Befehl RTI zu verlassen. Noch ein letzter Hinweis: Obwohl das A-Register beim Sprung auf IRQV1 in &FC gespeichert wurde, kann ein weiterer Interrupt durchaus veranlaßt haben, daß dieser Wert vor Beendigung Ihres eigenen Interrupts überschrieben wurde.

Interrupts des VIA-Chips

Flag-Register der Interrupts



Register zur Freigabe von Interrupts



Obwohl der 6502-Prozessor des Acorn B nur zwei Interruptleitungen – IRQ und NMI – besitzt, läßt sich der Schnittstellenchip zur Peripherie-VIA so programmieren, daß sich mehrere Systemkomponenten die gleiche IRQ-Interruptleitung des Prozessors teilen können. Jeder VIA-Chip hat zwei E/A-Ausgänge im Acht-Bit-Format, von denen jeder wiederum über acht Datenleitungen und je ein Paar Signalleitungen verfügt. Die Leitungen von Ausgang A heißen CA1 und CA2 und die von Ausgang B CB1 und CB2. Sie werden oft für das „Handshaking" (die Synchronisation der Datenübertragung zwischen Micro und einem Peripheriegerät) eingesetzt. Die Leitungen können aber auch als Interruptleitungen programmiert werden. Zusätzlich zu den vier möglichen Interruptleitungen jeder VIA gibt es ein internes Interrupt, das sich über das Shift-Register des Chips generieren läßt. Dieses Register steuert die serielle Datenübertragung, die von externen, über CB1 hereinkommenden Impulsen ausgelöst wird und über CB2 abläuft. Schließlich gibt es noch zwei Intervalltimer, die ebenfalls Interrupts erzeugen können. All diese Interruptquellen sind mit der einzigen IRQ-Leitung des Prozessors verbunden.

Zur Bearbeitung eines Interrupts muß der Prozessor die Quelle kennen. Die CPU fragt dazu in der VIA das Flag-Register für Interrupts ab. Jedes Bit dieses Registers entspricht einer der potentiellen Interruptquellen und wird auch von der Interruptanforderung dieser Systemkomponente gesetzt. Eine Ausnahme ist Bit 7, das nur gesetzt wird, wenn ein Peripheriegerät ein Interrupt anfordert.

In manchen Situationen muß verhindert werden, daß Interrupts ausgegeben werden, während der Prozessor gerade einen anderen Interrupt bearbeitet. Diese Information kann in das Register zur Freigabe von Interrupts („Interrupt Enable Register" – IER) programmiert werden.

Ereignisreich

Ein „Ereignis“ (englisch: Event) ist eine Art abschaltbares Interrupt. Wir untersuchen einige Ereignisse, die auf dem Acorn B zur Verfügung stehen.

Es gibt eine ganze Reihe von Situationen, in denen der Acorn B „Ereignissignale“ erzeugt: beim Eintrag von Zeichen in den Eingabebuffer, nach der Generierung eines „Vertical Sync Signals“, nach der Umwandlung eines Signals durch den A/D-Wandler und beim Drücken der Escape-Taste. Einige dieser Mechanismen erzeugen Interrupts, die dann über den IROV1-Vektor bearbeitet werden.

Mit diesen Ereignissen lassen sich daher – quasi durch die Hintertür – IRQs mit hoher Priorität einsetzen. Ereignisse müssen „angeschaltet“ werden, damit der Computer sie erzeugt und bearbeitet. Ist ein bestimmtes Ereignis angeschaltet, übergibt die CPU die Steuerung an eine Routine, deren Startadresse in dem Vektor bei &220 und &221 gespeichert ist. Um zu verhindern, daß diese Routine bei jeder Auslösung angesprochen wird, können Ereignisse auch „abgeschaltet“ werden. Das An- und Abschalten geschieht über zwei OSBYTE-Routinen. Die Tabelle zeigt die wichtigsten Ereignisse und die CALLs von OSBYTE, die sie an- und abschalten.

Befehle, die Ereignisse an- und abschalten

Ereignis	An	Aus
Ausgabebuffer wird entleert	*FX14,0	*FX13,0
Eingabebuffer wird gefüllt	*FX14,1	*FX13,1
Zeichen erreicht Eingabebuffer	*FX14,2	*FX13,2
A/D-Umwandlung beendet	*FX14,3	*FX13,3
Start des Vertical Sync	*FX14,4	*FX13,4
Intervalltimer ist abgelaufen	*FX14,5	*FX13,5
Escape wurde gedrückt	*FX14,6	*FX13,6

Um beispielsweise das Ereignis „Interval-Timer ist abgelaufen“ auszulösen, muß in BASIC nur *FX14,5 ausgeführt werden (oder der entsprechende Maschinencodebefehl). Nach Anschalten des Ereignisses löst der Timer jedesmal, wenn er Null erreicht, die Routine aus, deren Adresse im Ereignisvektor gespeichert ist. Das Programm stürzt ab, wenn sich dort kein Maschinencodeprogramm befindet.

Sie haben sicher bemerkt, daß es nur einen Ereignisvektor gibt, aber eine ganze Reihe von Ereignissen. Alle angeschalteten Ereignisse sprechen die gleiche Routine an, die jedes mögliche Ereignis bearbeitet. Wie weiß nun die Routine, von welcher Quelle sie ausgelöst wurde? Wenn nur ein Ereignis angeschaltet ist, ist die Situation eindeutig, bei mehreren gibt der Inhalt des A-Registers beim Einsprung in die Routine Auskunft über die Quelle. Die Tabelle zeigt den Registerinhalt beim Einsprung in die Routine, auf die der Ereignisvektor zeigt.

Registerinhalt beim Eintritt in die Ereignisroutinen

Ereignis	A-Register	X-Register	Y-Register
Ausgabebuffer	0	Buffernummer	#
Eingabebuffer wird entleert	1	Buffernummer	#
Zeichen aus Eingabebuffer	2	#	Zeichen
A/D-Umwandlung	3	#	Kanalnummer
Start des Vertical Sync	4	#	#
Intervalltimer	5	#	#
Escape wurde gedrückt	6	#	#

Bei einem # ist der Registerinhalt nicht festgelegt. Das bedeutet, daß die Ereignisroutine anfangs den Wert des A-Registers überprüfen kann und so feststellt, welches Ereignis die Routine ausgelöst hat.

Wenn Sie eine Routine zur Bearbeitung eines Ereignisses schreiben, müssen Sie zuerst den Registerinhalt sichern, da die entsprechende OS-Routine des Acorn B nicht wie bei Interrupts den Inhalt des A-Registers und das Status-Register auf den Stapel schiebt. Weiterhin gibt diese OS-Routine die Steuerung mit RTS an das unterbrochene Programm zurück. Doch genug der Theorie, sehen wir uns einige Beispiele an.

Große Ereignisse

- **Ereignis 0** wird beim Leeren des Ausgabebuffers erzeugt. Die Buffernummer ist die gleiche wie bei OSBYTE 21.
- **Ereignis 1** wird erzeugt, wenn im aktuellen Eingabebuffer ein weiteres Zeichen gespeichert werden soll. Auch hier entspricht die Buffernummer OSBYTE 21. Dabei ist der ASCII-Code, der nicht in den Buffer geschrieben werden konnte, beim Ansprechen der OS-Routine im Y-Register gespeichert.
- **Ereignis 2** wird erzeugt, wenn ein Zeichen den Eingabebuffer erreicht – normalerweise nach einem Tastendruck. Das Programm zeigt die Grundelemente einer Ereignisroutine, die bei jedem Tastendruck ein VDU 7 erzeugt. Das Ereignis tritt auch während des Ablaufs eines BASIC-Programms ein.

```
10 DIM mc% (100)
20 FOR I%=0 TO 2 STEP 2
30 P%=mc%
40 [OPT I%
50 PHP
60 PHA
70 TXA:PHA
80 TYA:PHA
90
100 LDA#7
110 JSR&FFE3
120
130 PLA:TAY
140 PLA:TAX
150 PLA
160 PLP
170 RTS
180
190 ]:NEXT I%
200 ?&220=mc%MOD256:REM niederwertiges Byte der im Vektor
    gespeicherten Adresse
210 ?&221=mc%DIV 256:REM höherwertiges Byte der Adresse
220 *FX14,2
230 REM das Ereignis nur anschalten,
    nachdem
240 REM die Routine angelegt wurde
250 REPEAT
260 PRINT I%
270 I%=I%+1
280 UNTIL FALSE
```

● **Ereignis 3** wird erzeugt, wenn einer der vier Kanäle des A/D-Wandlers im Acorn B eine Umwandlung vollendet hat. Je nach Wandlungsrate findet das Ereignis alle fünf oder zehn Millisekunden statt.

● **Ereignis 4** wird am Anfang eines Vertical Sync Impulses generiert. Es tritt pro Sekunde 50mal auf und kann somit gut als Quelle für Zeitimpulse dienen.

● **Ereignis 5:** Wie die Variable TIME wird auch der Intervalltimer jede Hundertstelsekunde aktualisiert. Dieser Timer wurde schon bei der Behandlung von OSWORD erwähnt. Er wird inkrementiert und löst beim Erreichen von Null ein Ereignis aus. Normale Abläufe lassen sich damit leicht in regelmäßigen Abständen unterbrechen. Über OSWORD mit A=4 wird in den Intervalltimer geschrieben und über OSWORD mit A=3 daraus gelesen.

Um im Acorn B nach einer Sekunde ein Ereignis auszulösen, wird der Intervalltimer mit seinem Maximalwert −100 geladen. Nach 100 Inkrementierungen erreicht der Timer dann den Wert Null. Der Timer hat eine Kapazität von fünf Bytes und somit den Maximalwert &FFFFFFFFFF. Das Programm zeigt diesen Vorgang und erzeugt dabei in regelmäßigen Abständen einen Signalton:

```
10 DIM mc% (100), delay%10
20 FOR I%=0 TO 2 STEP 2
30 P%=mc%
40 [OPT I%
50 PHP:PHA
60 TXA:PHA
70 TYA:PHA
80 LDA#7:JSR &FFE3
90 .clock set/reset für das nächste Interrupt
100 LDX# delay% MOD 256/ OSWORD
    anlegen
110 LDY #delay%DIV 256
120 LDA #4
130 JSR&FFF1
140 PLA:TAY
150 PLA:TAX
160 PLA:PLP
170 RTS
180 ]:NEXT I%
190 ?&220=mc%MOD 256
200 ?&221=mc%DIV 256
210 !delay%=&FFFFFF9C:delay%?4=&FF
220 *FX14,5
230 CALL clock set:REM clock starten
240 END
```

● **Ereignis 6** wird durch Drücken der Escape-Taste ausgelöst.

Es gibt drei weitere Ereignisse, die entweder einen Fehler des RS423-Systems, einen Econet-Fehler oder ein vom Anwender programmiertes Ereignis anzeigen. Unsere nächste Folge enthält einen zusammenfassenden Überblick über das OS des Acorn B.

Testen Sie Ihre Kenntnisse

Das folgende Programm enthält Mechanismen, die in der Serie über das Betriebssystem des Acorn B erklärt wurden. Sehen Sie sich das Programm genau an, geben Sie es ein und lassen Sie es ablaufen. Versuchen Sie dabei herauszufinden, was das Programm bewirkt und wie es arbeitet. In der nächsten Folge bringen wir eine ausführliche Beschreibung und weitere Programmbeispiele.

```
  20 PROCassemble
  30 END
  50 DEFPROCassemble
  70 DIM MC% &FF
  80 oldvec=?&20E+256*?&20F
  90 FOR pass=0 TO 3 STEP 3
 100 P%=MC%
 120 [OPT pass
 130.temp EQUB 00
 140.start
 150 PHP
 160 STA temp
 170 TXA:PHA
 180 TYA:PHA
 200 JSR check
 230 PLA:TAY
 240 PLA:TAX
 250 LDA temp
 260 PLP
 270 JMP oldvec
 290 .check
 300 LDA temp
 310 CMP#65
 320 BMI out
 330 CMP #91
 340 BPL out
 350 ADC #32
 360 STA temp
 370 .out RTS
 390 ]:NEXT
 400 ?&20E=start MOD 256
 410 ?&20F=start DIV 256
 420 ENDPROC
```

Letzter Aufruf

Bei unserer Untersuchung des Betriebssystems des Acorn B legten wir dar, daß ROM-Routinen in bestimmten Situationen die Reaktion des OS stark beeinflussen. Wir beschließen diese Serie mit einigen Programmbeispielen aus der Praxis.

OSWRCH hat unter anderem die Funktion, BASIC-Programmlisten auf den Bildschirm zu bringen. Unser Maschinencode prüft nun den ASCII-Code jedes Zeichens, das während der OSWRCH-Routine im Akkumulator auftaucht. Liegt der ASCII-Wert zwischen 64 und 91 (Großbuchstabe), dann wird für den entsprechenden Kleinbuchstaben 32 addiert. Nach Ablauf unseres Codes geht die Steuerung mit einem Sprung (JMP) auf den ursprünglichen OSWRCHV-Vektor wieder an die eigentliche OSWRCH-Routine zurück. Alle Großbuchstaben der Listings erscheinen nun als Kleinbuchstaben auf dem Bildschirm. Die BREAK-Taste setzt OSWRCHV wieder auf den alten Wert.

Das erste Listing zeigt ein Dienstprogramm, das während der Programmierung die Speicherkapazität beobachtet. Der Ausdruck

```
HIMEM-(?2+256*?3)
```

liefert die Bytezahl, die noch zur Verfügung steht (Speicherstelle 2 und 3 enthalten die Adresse der Obergrenze der BASIC-Variablentabelle). Da es lästig ist, diesen Wert ständig abzufragen, arbeitet die Dienstroutine mit dem Ereignis „Zeicheneingabe in den Eingabebuffer" und bewertet bei jedem Tastendruck automatisch den obenstehenden Ausdruck. Fällt der verbleibende freie Speicher unter einen bestimmten Wert, erklingt ein Signalton. Die Routine wird bei jedem Tastendruck aktiviert und läuft beim Eingeben eines Programms ständig im Hintergrund ab.

Begründete Schwellenangst

Zunächst fragt das Programm nach einem Schwellenwert, von dem an es melden soll, daß Speicherkapazität knapp wird. Das Modul PROCselect-memory bei Speicherstelle &0A00 legt diesen Schwellenwert im lo-hi Format fest. Es schiebt zunächst die Registerinhalte zur Sicherung auf den Stapel. Nach Ablauf der Subroutine werden die Register zurückgeladen, während RTS die Steuerung an das Programm zurückgibt, das das Ereignis ausgelöst hatte. Nun geht es ins Detail.

Das eigentliche Submodul ruft zwei weitere Subroutinen auf, die den verbleibenden Speicher berechnen. Die Routine „inc" nimmt den in Speicherstelle 2 und 3 abgelegten Wert, addiert die nieder- und höherwertigen Bytes des Schwellenwertes und speichert das Ergebnis in &70 und &71. Die Routine „sub" zieht dann mit einer 16-Bit-Berechnung den Inhalt von &70 und &71 von HIMEM ab. Der Wert von HIMEM befindet sich in den Speicherstellen &06 und &07 der Zero Page. Wir dürfen hier absolute Speicheradressen verwenden.

Nach Ausführung der Subtraktion prüft Zeile 490 das Vorzeichen des Ergebnisses. Bei einem positiven Wert sind mehr Bytes frei, als der Schwellenwert angibt. Ein negatives Ergebnis dagegen löst über OSWRCH mit A=7 einen Signalton aus, und zeigt damit an, daß der Speicherplatz ab dieser Stelle im Programm nicht mehr genügt.

Speicherüberprüfung

```
100 REM **** Using Key Pressed Event ****
110 REM **** To keep an exe on memory use ****
120 MODE 1
130 PROCselect_memory
140 PROCassemble
150 END
160
170 DEFPROCselect_memory
180 INPUT"Beep when how many % left",n:n=n*1024
190 hibyte=n DIV 256
200 lobyte=n MOD 256
210 ENDPROC
220
230 DEFPROCassemble
240 oswrch=&FFEE
250
260
270 FOR pass=0 TO 3 STEP 3
280 P%=&0A00
290
300 [OPT pass
310 .start
320 PHP
330 PHA
340 TXA:PHA
350 TYA:PHA
360
370 JSR memory_check
380
390
400 PLA:TAY
410 PLA:TAX
420 PLA
430 PLP
440 RTS
450
460 .memory_check
470 JSR inc
480 JSR sub
490 BPL room
500 LDA #7
510 JSR oswrch
520 .room RTS
530
540 .sub SEC
550 LDA &06:SBC &70:STA &70
560 LDA &07:SBC &71:STA &71
570 RTS
580 .inc CLC
590 LDA &02:ADC #lobyte:STA &70
600 LDA &03:ADC #hibyte:STA &71
610 RTS
620
630 ]:NEXT
640 ?&220=start MOD 256
650 ?&221=start DIV 256
660 *FX14,2
670 ENDPROC
```

Unser nächstes Programm löst bei dem Ereignis „Intervalltimer ist abgelaufen" eine Tonfolge aus, während der Computer an anderen Aufgaben weiterarbeitet. Da die Ereignisroutine im Hintergrund abläuft, kann so das Sichern, Editieren, Drucken und der Ablauf eines anderen Programms mit Musik untermalt werden. Sie könnten mit diesem Modul aber auch Spiele mit Melodien unterlegen, die beispielsweise beim Laden von Dateien erklingen, bis der Vorgang abgeschlossen ist.

Der Programmstart liegt bei &0C00. Zuerst wird eine Initialisierungsroutine mit folgenden Aufgaben ausgeführt: Der Ereignisvektor EVENTV bei &220 und &221 wird auf unsere Routine umgestellt. Danach aktiviert OSBYTE 14 mit X=5 das Ereignis „Intervalltimer ist abgelaufen" an. Ein „Notenzähler" bei &70 wird auf Null gestellt und schließlich der Ereignistimer mit einem Aufruf der Subroutine „Clock" gestartet.

„Soundtable"

Die eigentliche Routine schiebt zunächst die CPU-Register auf den Stapel und ruft dann eine Subroutine auf, die eine einzelne Note spielt. Danach wird der Timer wieder auf Null gesetzt, und schließlich werden die Register zurückgeladen. Beachten Sie, daß alle Ereignisroutinen die Register standardmäßig speichern und wiederherstellen. Es muß daher nur der Code zwischen diesen beiden Programmblöcken verändert werden.

Jedesmal wenn der Ereignistimer die Null erreicht, ertönt eine aus der Notentabelle „notetable" ausgewählte Note. Diese Tabelle besteht nur aus einer Serie von Bytes, die die Tonhöhe der gewünschten Töne festlegen. Ein Zähler in &70 bestimmt, welcher der fünf Töne gespielt werden soll. Der Tonhöhenwert wird nun gelesen und auf dem entsprechenden Platz von „soundtable" abgelegt. „Soundtable" ist der Parameterblock für die OSWORD-Routine, mit der wir später den Ton erzeugen. Danach wird der Zähler inkrementiert, der nun auf die nächste Position der Notentabelle zeigt. Da sich in der Tabelle nur fünf Werte befinden, wird der Zählerwert in &70 wieder auf Null gesetzt, wenn er die Fünf erreicht hat.

Die Subroutine „clock" aktiviert den Ereignistimer über OSWORD mit A=4. Im Speicherbereich „time" befindet sich der Wert für das Timerregister, das die Pausen zwischen den einzelnen Tönen festlegt. Die Subroutine „sound" arbeitet (wie der BASIC-Befehl SOUND) mit OSWORD mit A=7, wobei „soundtable" den Parameterblock des OSWORD-Aufrufs darstellt. Die BASIC-Befehle in den Zeilen 840 und 850 legen die Daten für „notetable" und „soundtable" fest.

BREAK setzt den Inhalt von EVENTV wieder auf seine ursprünglichen Werte und stoppt so das Programm. Der Aufruf von „initialise" startet die Routine dann wieder neu.

Backgroundmusik

```
10 REM **** SOUND EVENTS ****
20
30 PROCassemble
40 CALL initialise
50 END
100
110 DEFPROCassemble
120 note_count=&70
130 osword=&FFF1
140 osbyte=&FFF4
150 oswrch=&FFEE
160
170 FOR pass=0 TO 2 STEP 2
180 P%=&0C00
190
200 [OPT pass
210 .initialise
220 LDA #event MOD 256:STA &220
230 LDA #event DIV 256:STA &221
240 LDA #14:LDX #5:JSR osbyte
250 LDA #0:STA note_count
260 JSR clock
270 RTS
280
290 .event
300 PHP
310 PHA
320 TXA:PHA
330 TYA:PHA
340
350 JSR play_note
360 JSR clock
370
380
390 PLA:TAY
400 PLA:TAX
410 PLA
420 PLP
430 RTS
440
450 .play_note
460 LDY note_count
470 .sloop LDA notetable,Y
480 STA soundtable+4
490 JSR sound
500 INC note_count
510 LDA note_count:CMP#5:BNE out
520 LDA #0:STA note_count
530 .out RTS
540
550
560 .clock
570 LDX #time MOD 256
580 LDY #time DIV 256
590 LDA #4
600 JSR osword
610 RTS
620
630 .sound PHA
640 TXA:PHA
650 TYA:PHA
660 LDX #soundtable MOD 256
670 LDY #soundtable DIV 256
680 LDA #7
690 JSR osword
700 PLA:TAY
710 PLA:TAX
720 PLA
730 RTS
740
750 .notetable
760 EQUD &00000000
770 EQUD &00000000
780 .soundtable
790 EQUD &00000000
800 EQUD &00000000
810 .time EQUD &FFFFFFCF
820 EQUB &FF
830 ]:NEXT
840 FOR I%=0 TO 4:READ data?
    (notetable+I%)=data:NEXT
850 FOR I%=0 TO 2:READ data?
    (soundtable+I%)=data:NEXT
860 ENDPROC
870 DATA 89,73,81,99,97
880 DATA 1,0,&FA,&FF,0,0,10,0
```

Unser letztes Programm zeigt, wie Sie mit Ereignissen einen einfachen grafischen Umriß (in diesem Fall ein Rechteck) bei jedem Eintreten des Ereignisses auf dem Schirm um eine Position verschieben können. Die Geschwindigkeit der Bewegung hängt vom Wert des Ereignistimers ab, der in Zeile 1260 gesetzt wird. Die Definition der zu bewegenden Form ist mit PLOTzahlen in „shapetable" gespeichert. „Shapetable" wird von einem DATA-Befehl definiert und in Zeile 1220 mit POKE in den Speicher gesetzt. Zeile 1230 und 1250 setzen EVENTV auf unser Programm und schalten das entsprechende Ereignis an.

Subroutinenaufrufe

Zeile 200 bis 230 sichern die Register, während die Subroutinenaufrufe der Zeilen 250 bis 280 die eigentliche Arbeit ausführen. Die Register werden schließlich wiederhergestellt und ein RTS ausgeführt. Die Zeilen 370 bis 440 sichern den aktuellen Status des Grafikcursors und von GCOL, um sie dann schließlich nach dem Rücksprung von der Ereignisroutine zurückladen zu können.

Die Zeilen 460 bis 520 bewegen den Umriß, und Zeile 540 bis 630 versetzen die Grafikfarbe und den Grafikcursor wieder in den Zustand vor dem Ansprung der Routine.

Die Zeilen 650 bis 720 führen mit der OSWRCH-Routine, die dem BASIC-Befehl MOVE entspricht, die Bewegung zu den gewünschten X- und Y-Positionen aus. Diese Positionen werden in xpos und xpos+1 für die

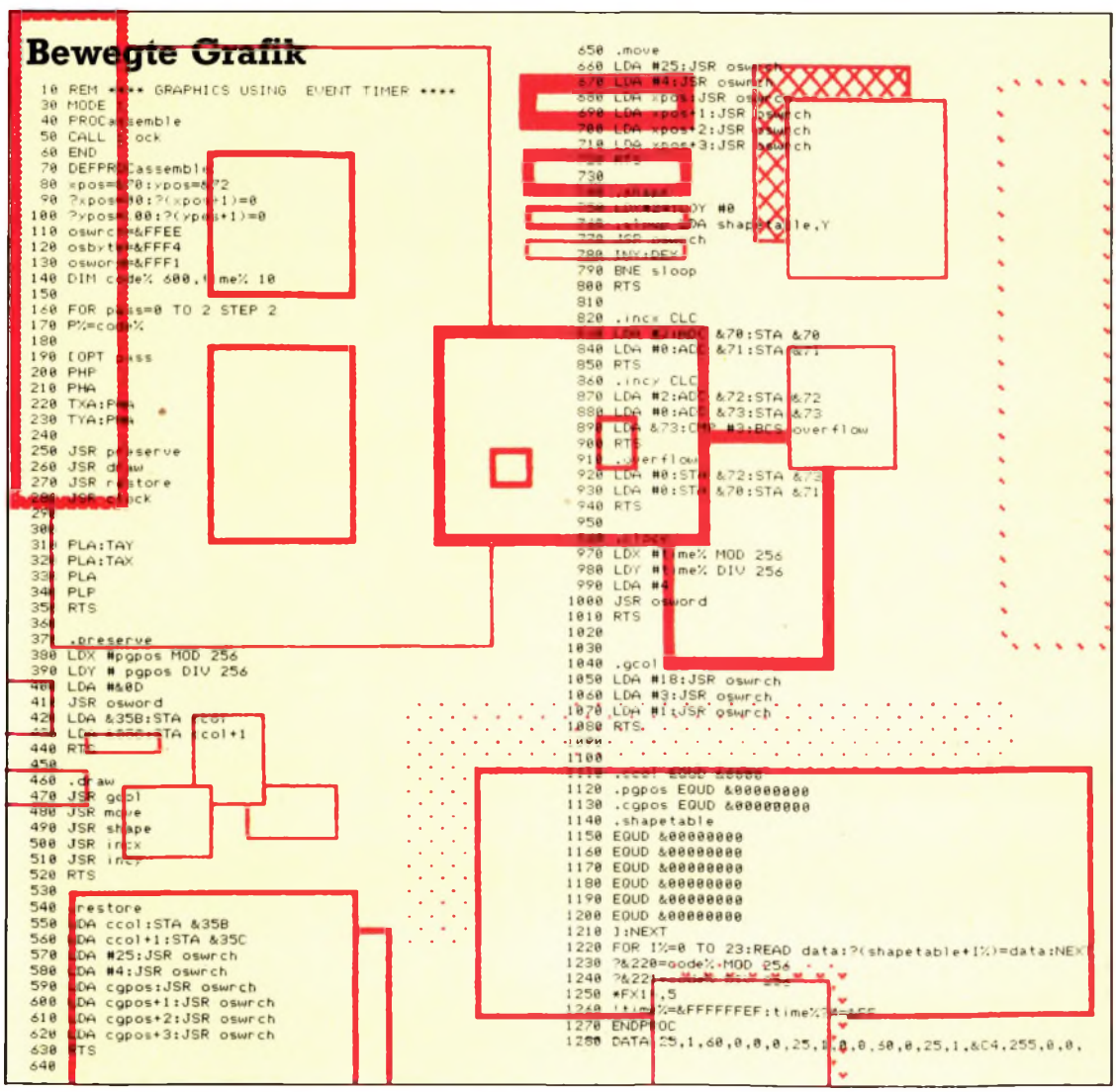
X-Koordinate und ypos und ypos+1 für die Y-Koordinate festgelegt. Die Zeilen 740 bis 800 schließlich zeichnen – wiederum mit OSWRCH – den Umriß. Das Y-Register dient dabei als Indexregister für die Bytetabelle „shapetable", in der sich der Umriß befindet.

Bei der Bewegung des Umrisses über den Schirm wird deutlich, daß die Positionen von x und y bei jedem Eintritt des Ereignisses aktualisiert werden müssen. Die Routinen „incx" und „incy" in Zeile 820 bis 900 führen diese Aufgabe aus. Sobald das Rechteck die obere Bildschirmgrenze erreicht, stellt die Routine „overflow" die X- und Y-Koordinaten auf Null zurück. Die clock-Routine schaltet über OSWORD mit A=4 den Ereignistimer aus, wobei „gcol" dem BASIC-Befehl GCOL3,1 entspricht.

Erhöhter Zeitwert

Das Programm ist nicht ganz ohne Probleme. Solange das Scrollen noch nicht eingesetzt hat, bewegt sich der Umriß recht unsicher über den Bildschirm, während Sie andere Vorgänge ausführen können, die dadurch nicht gestört werden. Wenn Sie die Bewegung jedoch beschleunigen, indem Sie den Wert von „time" erhöhen, funktioniert z. B. das Listen eines Programms bei der schnelleren Bewegung des Rechtecks zwar immer noch, doch tauchen seltsame Zeichen auf dem Schirm auf. Zwischen unserer Routine, die bei jedem Ereignis ausgelöst wird, und den normalen OS-Abläufen eines Listings entsteht ein Konflikt. Da beide Routinen mit OSWRCH arbeiten, findet die Listroutine nach der Unterbrechung durch unsere Ereignisroutine OSWRCH nicht mehr in dem Zustand vor, den sie erwartet.

Wenn beim Unterbrechen von OSWRCH die unterbrechende Routine wiederum OSWRCH aufruft, kann OSWRCH nur einmal verlassen werden. Die Firma Acorn schlägt daher vor, OS-Routinen nicht in Ereignis- und Interruptroutinen zu verwenden. Sie lassen sich zwar einsetzen, doch sollten Sie dabei sehr vorsichtig sein. Wenn bei Interrupts keine OS-Routinen verwandt werden, entstehen auch keine Probleme. Hier haben wir eine der wenigen Situationen, in denen der Einsatz von ROM-Routinen nicht zu empfehlen ist. Grafik und Daten lassen sich aber auch direkt in den Bildschirmspeicher schreiben.



Der Prozessor 6510

Das Betriebssystem des Commodore 64 hat viele praktische, aber auch einige ungewöhnliche Aspekte. Wir betreiben eine Bildschirmuhr.

Systemprogrammierer müssen die „Memory Map", das heißt den Aufbau des Arbeitsspeichers, kennen. Der Microprozessor 6510 des Commodore 64 kann – je nach Konfiguration der Maschine – mit einer von mehreren Speicherstrukturen arbeiten, da er mit der Technik des „Bank Switching" Zugriff auf unterschiedliche Speicherblocks hat. Die Umschaltung geschieht dabei entweder über die Hardware (an die Kontakte 8 und 9 der Erweiterungsbuchse wird hohe (+5V) oder niedrige (0V) Spannung angelegt) oder über die Software (durch die Änderung des Inhalts der Adresse 1). Beide Methoden ändern radikal die Interpretation des Speicheraufbaus. Die Tabelle gibt einen Einblick in den standardmäßig vorgegebenen Speicheraufbau der 64 KBytes, die der 6510 ansprechen kann.

Wenn wir davon ausgehen, daß der Commodore 64 mit der Standardeinstellung arbeitet, dann fällt als erstes auf, daß der Computer 64 KByte RAM und 20 KByte ROM enthält. Da der 6510 von diesen 84 KBytes aber nie mehr als 64 adressieren kann, wird hier das Bank Switching eingesetzt. Die Memory Map zeigt, daß sich das ROM des BASIC-Interpreters zwischen $A000 und $CFFF befindet und das ROM für die Ein- und Ausgabefunktionen zwischen $E000 und $FFFF.

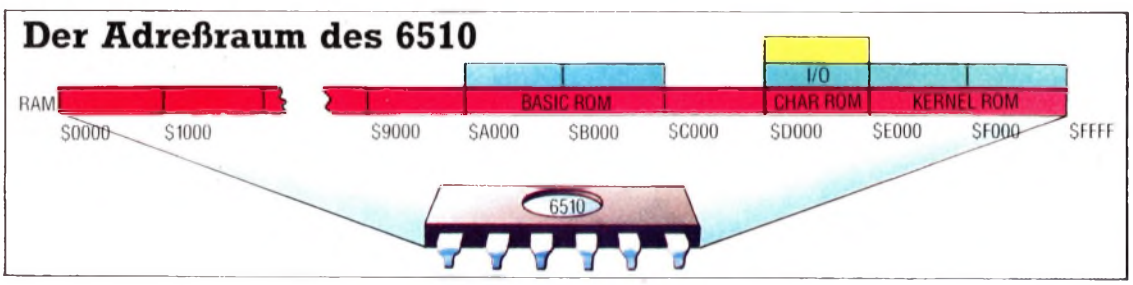
Schattendasein

Unter diesen beiden ROM-Blöcken verbergen sich zwei RAM-Blocks zu je acht KByte, die der 6510 normalerweise nicht ansprechen kann (ein POKE oder STA auf diese Adressen könnte dahin vordringen, PEEK liefert jedoch immer nur den Inhalt des darüberliegenden ROMs). Das RAM läßt sich lesen, wenn das entsprechende ROM mit Bank Switching ausgeschaltet wird. Denken Sie beim Einsatz der BASIC-Befehle PEEK und POKE jedoch daran, daß nicht auch das BASIC-ROM ausgeschaltet wird.

Wie jedes BASIC-Programm brauchen auch der Interpreter und die E/A-Routinen einen RAM-Bereich (das OS-RAM) zur Speicherung von Variablen. Das OS-RAM reicht von $0002 bis $03FF, wobei ein wichtiger Teilbereich – der Stack – zwischen $0100 und $01FF liegt.

Eile ist geboten

Das OS-RAM liegt auf der Zero Page, da dieser Speicherbereich den schnellsten RAM Zugriff bietet und die Geschwindigkeit der OS-Routinen außerordentlich wichtig ist. Wenn Sie in einem BASIC-Programm Maschinencoderoutinen einsetzen, sollten Sie diesen RAM-Bereich daher genau kennen. Sehen wir uns zwei wichtige Aspekte des OS-RAM genauer an: die BASIC-Pointer und überaus nützlichen Maschinencodevektoren.



Der 6510 Prozessor arbeitet mit 16-Bit-Adressen und kann daher maximal auf 65536 (64K) Speicherstellen zugreifen. Die insgesamt 84K (RAM und ROM) des Commodore 64 werden mit der Technik des „Bank Switching" adressiert, die bestimmte Adreßbereiche in der „Memory Map" ein- oder ausschaltet.

Der vordere „Block" zeigt, wie der 6510 seinen Speicher normalerweise interpretiert. Wenn spezielle „Banking"-Register gesetzt sind, lassen sich aber auch die darunterliegenden Bereiche ansprechen.

Die Memory Map des Commodore 64

Adressen	Funktionen
$0000 –$0001	Steuerregister der Memory Map des 6510
$0002 –$03FF	Betriebssystem-RAM
$0400 –$07F7	Bildschirm-RAM
$07F8 –$07FF	Sprite Pointer
$0800 –$9FFF	BASIC-Programmbereich (mit Variablen)
$A000 –$BFFF	ROM des BASIC-Interpreters
$C000 –$CFFF	4 KByte freies RAM
$D000 –$D02E	VIC II (6566/9) Chip-Steuerregister
$D02F –$D3FF	Wiederholung der VIV II Abbildung
$D400 –$D41C	SID (6581) Chip-Steuerregister
$D41D –$D7FF	Wiederholung der SID-Abbildung
$D800 –$DBE7	Farbnybbles (je 4 Bits)
$DBE8 –$DBFF	Nicht eingesetzte Nybbles
$DC00 –$DC0F	CIA # 1 E/A (6526) Chip-Steuerregister
$DC10 –$DCFF	Wiederholung der CIA # 1 Abbildung
$DD00 –$DD0F	CIA # 2 E/A (6526) Chip-Steuerregister
$DD10 –$DDFF	Wiederholung der CIA # 2 Abbildung
$E000 –$FFFF	ROM für Ein- und Ausgabe

Pointertabelle des C64 BASIC

Adressen	Pointer
$002B—$002C	Anfang des BASIC-Bereichs für Anwender
$002D—$002E	Anfang des Bereichs für BASIC-Variablen
$002F—$0030	Anfang des Bereichs für BASIC-Arrays
$0031—$0032	Ende der BASIC Arrays+1
$0033—$0034	Untergrenze für Strings
$0035—$0036	Dienstbereich
$0037—$0038	Ende des BASIC-Bereichs für Anwender

BASIC Pointer

Zuweilen kann eine Änderung der BASIC-Pointer sehr praktisch sein. Der normale Inhalt von $002B (dezimal 43) und $002C (dezimal 44) ist beispielsweise 1 und 8 und zeigt so die Anfangsadresse des BASIC an (im Format lo-hi). BASIC beginnt also bei 8*256+1=2049 oder $0801. Tatsächlich liegt der Anfang von BASIC zwar bei $0800, doch verlangt das Betriebssystem, daß das erste Byte auf Null steht. (Nebenbei: Wenn Sie mit dem Maschinencodemonitor des Commodore arbeiten, ist $0801 auch die Adresse, von der an Sie ein BASIC-Programm sichern müssen.)

Vor dem Laden von BASIC-Programmen können Sie mit diesen beiden Pointern die Untergrenze des BASIC verlegen. BASIC darf jedoch nur an einer Seitengrenze beginnen (das bedeutet, der Inhalt von $002B sollte auf Eins bleiben) und muß mit einer Null anfangen (denken Sie daran, daß eine „Seite" 256 Bytes enthält). Der direkte Befehl

POKE2560,0:POKE44,10:NEW

erhöht die Untergrenze des Speichers um zwei Seiten auf $2560. NEW ist die schnellste Methode, alle Pointer (zwischen $002D und $0038) zurückzusetzen. Durch die Erhöhung der BASIC-Untergrenze können sich zwei BASIC-Programme gleichzeitig im Speicher befinden. Sie laden dabei nur das erste Programm, erhöhen die BASIC-Untergrenze und laden dann das zweite Programm.

Oft wird auch die Obergrenze des Speichers herabgesetzt, um Platz für Maschinencodeprogramme zu schaffen. Die direkten Befehle

POKE56,159:POKE51,0:POKE52,159

verlegen die Obergrenze des BASIC-Speichers um eine Seite nach unten.

Ein RAM-Block, der auf diese Weise dem Zugriff von BASIC entzogen wurde, kann nun sicher für den Maschinencode eingesetzt werden, da das OS diesen Bereich nicht mehr für das Speichern von BASIC-Variablen verwenden kann.

Die RAM-Vektoren

Neben den BASIC-Pointern haben auch die RAM-Vektoren zwischen $0314 und $0333 große Bedeutung für den Maschinencodeprogrammierer. Ein RAM-Vektor ist einer zusätzlichen Weiche in einem Schienenstrang nicht unähnlich. Wenn ein Zug (oder der 6510 bei der Ausführung eines Programms) diesen Weg nimmt, wird er im Normalfall die Weiche passieren, ohne den Seitenpfad zu nehmen. Manchmal ist es jedoch praktisch, den Zug über die Seitenlinie zu leiten und zusätzliche Stationen passieren zu lassen, bevor er auf der normalen Route weiterfährt.

Nehmen wir als Beispiel den IRQ-Vektor (Interrupt ReQuest): Im normalen Operationsmodus des Commodore 64 veranlaßt einer der Timer des 6526 E/A-Chips jede Sechzigstelsekunde, daß die IRQ-Leitung des 6510 auf niedrige Spannung gesetzt wird. Nach Beendigung des gerade bearbeiteten Befehls reagiert der 6510 auf die IRQ-Anforderung und springt auf ein Dienstprogramm, dessen Code bei $FF48 liegt. Die IRQ-Behandlungsroutine prüft nun unter anderem, ob auf der Tastatur eine Taste gedrückt wurde.

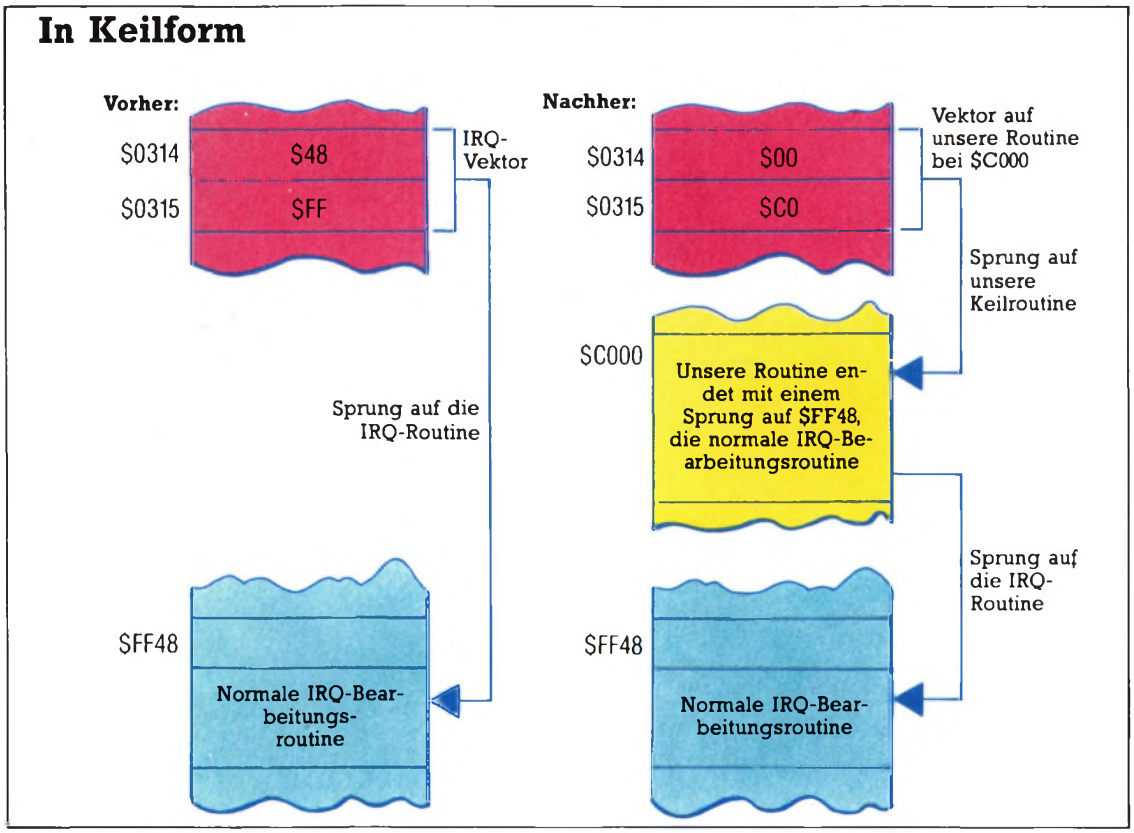
JMP ($0314)

ist einer der ersten Befehle der Routine. Er führt einen indirekten Sprung auf die Adresse aus, die in $0314 (niederwertiges Byte) und $0315 (höherwertiges Byte) gespeichert ist. Da diese beiden Bytes im RAM liegen, können wir ihren Inhalt ändern und den Vektor damit auf unseren eigenen Code zeigen lassen. Nach Ausführung unseres Codes wird der Microprozessor dann wieder auf das „Gleis" gesetzt, von dem wir ihn abgelenkt hatten. Wenn der Code nicht zu lang ist, kann der 6510 ihn jede Sechzigstelsekunde ausführen.

Vektorentabelle des C64

Adressen	Pointer
$0314 –$0315	IRQ Interrupt
$0316 –$0317	BRK Interrupt
$0318 –$0319	NMI Interrupt
$031A–$031B	E/A-Routine OPEN
$031C–$031D	E/A-Routine CLOSE
$031E–$031F	E/A-Routine CHKIN
$0320 –$0321	E/A-Routine CHKOUT
$0322 –$0323	E/A-Routine CLRCHN
$0324 –$0325	E/A-Routine CHRIN
$0326 –$0327	E/A-Routine CHROUT
$0328 –$0329	E/A-Routine STOP
$032A–$032B	E/A-Routine GETIN
$032C–$032D	E/A-Routine CLALL
$032E–$032F	Vom Anwender definiert
$0330 –$0331	E/A-Routine LOAD
$0332 –$0333	E/A-Routine SAVE

In Keilform



Der 6510 Prozessor wird jede Sechzigstelsekunde unterbrochen, um interne Organisationsaufgaben (das Abfragen der Tastatur auf Eingaben, etc.) auszuführen. Die Anfangsadresse dieser Bearbeitungsroutine liegt in dem Adreßpaar $0314 und $0315.

Wenn Sie den Inhalt dieser beiden Speicherstellen ändern, können Sie Ihren eigenen Code als „Keil" hineinschieben, der dann anstelle der normalen Bearbeitungsroutine ausgeführt wird. Normalerweise endet der Code eines derartigen „Keils" mit einem Sprung auf die normale IRQ-Bearbeitungsroutine. Sie stellen damit sicher, daß Ihr Code bei jedem Interrupt ausgeführt wird.

Die Zeit des Commodore 64

Das folgende Assemblerlisting und das Ladeprogramm in BASIC zeigen, wie ein „Keil" in die normale Bearbeitungsroutine für Interrupts eingesetzt wird. Die Keilroutine stellt am Bildschirmrand eine Uhr dar, die bei normalen Interrupts (die jede Sechzigstelsekunde eintreten) aktualisiert wird. Da die Uhr parallel zu anderen Vorgängen läuft, können Sie gleichzeitig mit BASIC-Programmen weiterarbeiten. Die Routine stellt zunächst die übergebene Anfangszeit ein und setzt dann den Inhalt von $0314 und $0315 auf den Anfang des „Keil"-Codes. Der Code aktualisiert die Zeit bei jedem IRQ-Interrupt. Letzter Befehl des Codes ist ein Sprung (JMP) auf die normale IRQ-Bearbeitungsroutine, deren Adresse in den Speicherstellen VECTOR und VECTOR+1 abgelegt wurde.

Statt den Quellentext einzugeben und zu assemblieren, können Sie das Programm auch mit DATA-Befehlen schreiben. Geben Sie dazu einfach das Ladeprogramm in BASIC ein und starten Sie es. Die Uhr erscheint am rechten oberen Bildschirmrand. Über eine Prüfsumme wird festgestellt, ob alle DATAs korrekt eingegeben wurden. Wenn das Programm mit der Meldung „CHECKSUM ERROR" abbricht, sollten Sie die DATAs überprüfen.

Die Uhr kann mit POKEs in die Speicherstellen 50129 (Stunden), 50130 (Minuten) und 50131 (Sekunden) gestellt werden. Im Ladeprogramm wird die Uhr von Zeile 1480 an auf den Anfangswert 4:30h gestellt. Sie können andere Zeiten einsetzen, wenn Sie die entsprechenden POKE-Werte ändern. SYS 50138 veranlaßt die Ausführung des Maschinencodes und setzt die Uhr in Gang. Die Uhr schaltet ab, wenn Sie die Tasten RUN/STOP und RESTORE gleichzeitig drücken, oder die spezielle Abschaltroutine im Maschinencodeprogramm mit SYS 50237 aktivieren.

Ladeprogramm in BASIC

```
1000 REM ** CLOCK BASIC LOADER **
1010 DATA173,166,2,240,10,169,128,13,14
1020 DATA221,141,14,221,48,8,169,127,45
1030 DATA14,221,141,14,221,169,127,45
1040 DATA15,221,141,15,221,173,208,195
1050 DATA41,128,141,208,195,173,209,195
1060 DATA32,28,197,13,208,135,141,11
1070 DATA221,173,210,195,32,28,197,141
1080 DATA10,221,173,211,135,32,28,197
1090 DATA141,9,221,169,0,141,8,221,120
1100 DATA173,20,3,141,214,195,173,21,3
1110 DATA141,215,195,169,76,141,20,3
1120 DATA169,196,141,21,3,88,96,120,173
1130 DATA214,195,141,20,3,173,215,195
1140 DATA141,21,3,88,96,173,216,195,201
1150 DATA6,240,3,76,8,197,169,255,141
1160 DATA216,195,173,213,195,240,243
1170 DATA173,11,221,170,41,128,208,5
1180 DATA169,1,76,111,196,169,16,141,38
1190 DATA4,173,212,195,141,38,216,163
1200 DATA13,141,39,4,173,212,195,141,39
1210 DATA216,138,41,16,32,14,197,141,28
1220 DATA4,173,212,195,141,28,216,138
1230 DATA32,22,197,141,29,4,173,212,195
1240 DATA141,29,216,169,58,141,30,4,173
1250 DATA212,195,141,30,216,173,10,221
1260 DATA170,32,14,197,141,31,4,173,212
1270 DATA195,141,31,216,138,32,22,197
1280 DATA141,32,4,173,212,195,141,32
1290 DATA216,169,47,141,33,4,173,212
1300 DATA195,141,33,216,173,9,221,170
1310 DATA32,14,197,141,34,4,173,212,195
```

```
1320 DATA141,34,216,138,32,22,197,141
1330 DATA35,4,173,212,195,141,35,216
1340 DATA169,46,141,36,4,173,212,195
1350 DATA141,36,216,173,8,221,105,48
1360 DATA141,37,4,173,212,195,141,37
1370 DATA216,238,216,195,108,214,195,74
1380 DATA74,74,74,24,105,48,96,41,15,24
1390 DATA105,48,96,160,255,56,200,233
1400 DATA10,176,251,105,10,141,217,195
1410 DATA152,10,10,10,10,13,217,195,96
1420 DATA42131:REM*CHECKSUM*
1430 CC=0
1440 FORI=50138TO50481
1450 READX:CC=CC+X:POKEI,X
1460 NEXT
1470 READX:IFCC<>XTHENPRINT"CHECKSUM ERROR"
1480 REM** TEST CLOCK **
1490 POKE50128,128:REM AM/PM
1500 POKE50132,8:REM COLOUR
1510 POKE50133,1:REM DISPLAY
1520 POKE50129,4:REM HOURS
1530 POKE50130,30:REM MINUTES
1540 POKE50131,0:REM SECONDS
1550 SYS50138:REM CALL ROUTINE

+0 TURN OFF THE WEDGE USE SYS50237
IN DIRECT OR PROGRAM MODE.
```

Assemblerlisting

```
;*************************************
;*                                   *
;*         CLOCK IRQ WEDGE           *
;*                                   *
;* 50128 = AM=0 / PM=128             *
;* 50129 = HOURS                     *
;* 50130 = MINUTES                   *
;* 50131 = SECONDS                   *
;* 50132 = CLOCK COLOUR              *
;* 50133 = DISPLAY ON=1/OFF=0        *
;*                                   *
;* WEDGE INSERT  SYS 50138           *
;* WEDGE REMOVE  SYS 50237           *
;*                                   *
;*************************************

IROVEC = $0314   ; IRQ RAM VECTOR
CLOCK = $DD08    ; TOD REGISTER
D2CRA = $DD0E    ; VIA#2 CRA
D2CRB = $DD0F    ; VIA#2 CRB
PALNTS = $02A6   ; PAL/NTSC FLAG
RATE = $06       ; DISPLAY EVERY 6 IRQS
DIGIT = $30      ; SCREEN CODE FOR '0'
POINT = $2E      ; SCREEN CODE FOR '.'
SLASH = $2F      ; SCREEN CODE FOR '/'
COLON = $3A      ; SCREEN CODE FOR ':'
HZ50 = $80       ; 50 HZ MASK FOR TODIN
HZ60 = $7F       ; 60 HZ MASK FOR TODIN
AY = $01         ; SCREEN CODE FOR 'A'
PEE = $10        ; SCREEN CODE FOR 'P'
EM = $0D         ; SCREEN CODE FOR 'M'
WRITE = 127      ; MASK TO SET CLOCK IN CRB
SCNLOC = $041C   ; CLOCK ADDR ON SCREEN
COLLOC = $D81C   ; ADDR ON VIDEO MATRIX
TRNCLO = $0F     ; MASK FOR LOW NYBBLE

* = $C3D0         ; START ADDRESS CODE
AMPM *=*+1       ; AM/PM FLAG
HOURS *=*+1      ; HOURS VAL (FOR INITIALISE)
MINS *=*+1       ; MINUTES VALUE
SECS *=*+1       ; SECONDS VALUE
COLOR *=*+1      ; CLOCK COLOUR
DISPLY *=*+1     ; DISPLAY/HIDE FLAG
VECTOR *=*+2     ; STORAGE FOR OLD IRQ VEC
COUNT *=*+1      ; IRQ COUNTER
TEMP1 *=*+1

;
;INSERT WEDGE
;

   LDA PALNTS       ; PAL OR NTSC
   BEQ NTSC         ; BRANCH FOR NTSC
   LDA #HZ50        ; MUST BE PAL
   ORA D2CRA
   STA D2CRA        ; SET TODIN FOR 50 HZ
   BMI PALDUN
NTSC
   LDA #HZ60        ; NTSC
   AND D2CRA
   STA D2CRA        ; SET TODIN FOR 60 HZ
PALDUN
   LDA #WRITE
   AND D2CRB        ; SET CLOCK NOT ALARM
   STA D2CRB
   LDA AMPM
   AND #128         ; MAKE AMPM VALUE VALID
   STA AMPM
   LDA HOURS        ; GET HOURS
   JSR BINBCD       ; CONVERT TO BCD
   ORA AMPM         ; OR WITH AM/PM FLAG
   STA CLOCK+3      ; STORE IN CLOCK
   LDA MINS         ; GET MINUTES
   JSR BINBCD       ; CONVERT TO BCD
   STA CLOCK+2      ; STORE IN CLOCK
   LDA SECS         ; GET SECONDS
   JSR BINBCD       ; CONVERT TO BCD
   STA CLOCK+1      ; STORE IN CLOCK
   LDA #00          ; ALWAYS SET 10THS TO 0
   STA CLOCK        ; START CLOCK
;
   SEI              ; DISABLE INTERRUPTS
   LDA IROVEC
   STA VECTOR       ; SAVE OLD IRQ VECTOR
   LDA IROVEC+1
   STA VECTOR+1
;
   LDA #<WEDGE
   STA IROVEC
   LDA #>WEDGE      ; INSERT WEDGE
   STA IROVEC+1
   CLI              ; ENABLE INTERRUPTS
   RTS

;
;REMOVE WEDGE
;
   SEI              ; DISABLE INTERRUPTS
   LDA VECTOR
   STA IROVEC       ; RESTORE RAM VECTOR
   LDA VECTOR+1
   STA IROVEC+1
   CLI              ; ENABLE INTERRUPTS
   RTS

;
;WEDGE STARTS HERE
;
WEDGE
   LDA COUNT        ;
   CMP #RATE        ; DO CLOCK THIS IRQ?
   BEQ CONT         ;
OUT
   JMP EXIT         ; NO
CONT
   LDA #$FF         ; RESET IRQ COUNTER
   STA COUNT
   LDA DISPLY       ; DISPLAY?
   BEQ OUT          ; NO...BRANCH

   LDA CLOCK+3      ; GET HOURS/AM/PM
   TAX              ; PUT A COPY IN X REG
   AND #$80         ; GET AM/PM
   BNE PM           ; BRANCH IF PM
   LDA #AY          ; DISPLAY 'A'
   JMP MERIOP
PM
   LDA #PEE         ; DISPLAY 'P'
MERIOP
   STA SCNLOC+10
   LDA COLOR        ; GET COLOUR
   STA COLLOC+10    ; SET COLOUR
   LDA #EM          ; DISPLAY 'M'
   STA SCNLOC+11
   LDA COLOR        ; SET COLOUR
   STA COLLOC+11
;
;DO HOURS
;
   TXA              ; GET HOURS
   AND #$10         ; JUST WANT HIGH DIGIT
   JSR HIDIGT       ; GET SCREEN CODE
   STA SCNLOC       ; DISPLAY IT
   LDA COLOR
   STA COLLOC       ; SET COLOUR
   TXA              ; GET BYTE AGAIN
   JSR LODIGT       ; GET LOW DIGIT
   STA SCNLOC+1     ; DISPLAY IT
   LDA COLOR
   STA COLLOC+1     ; SET COLOUR
;
   LDA #COLON       ; HRS/MINS SEPARATOR
   STA SCNLOC+2
   LDA COLOR
   STA COLLOC+2
;
;NOW DO MINUTES
;
   LDA CLOCK+2      ; GET MINUTES
   TAX
   JSR HIDIGT       ; DO HIGH DIGIT
   STA SCNLOC+3     ; DISPLAY IT
   LDA COLOR
   STA COLLOC+3     ; AND COLOUR
   TXA              ; GET BYTE AGAIN
   JSR LODIGT       ; DO LOW DIGIT
   STA SCNLOC+4     ; DISPLAY IT
   LDA COLOR
   STA COLLOC+4     ; SET COLOUR
;
   LDA #SLASH       ; MIN/SEC SEPARATOR
   STA SCNLOC+5
   LDA COLOR
   STA COLLOC+5
;
;NOW DO SECONDS
;
   LDA CLOCK+1      ; GET SECONDS
   TAX
   JSR HIDIGT       ; DO HIGH DIGIT
   STA SCNLOC+6     ; DISPLAY IT
   LDA COLOR
   STA COLLOC+6     ; AND COLOUR
   TXA              ; GET BYTE AGAIN
   JSR LODIGT       ; DO LOW DIGIT
   STA SCNLOC+7     ; DISPLAY IT
   LDA COLOR
   STA COLLOC+7     ; AND COLOUR
;
   LDA #POINT       ; SECS/TENTHS SEPARATOR
   STA SCNLOC+8
   LDA COLOR
   STA COLLOC+8
;
;NOW DO TENTHS
;
   LDA CLOCK        ; GET TENTHS VALUE
   ADC #DIGIT       ; ADD $30 FOR SCREEN CODE
   STA SCNLOC+9     ; DISPLAY IT
   LDA COLOR
   STA COLLOC+9     ; AND COLOUR
;
;
EXIT
   INC COUNT        ; INCREMENT IRQ COUNTER
   JMP (VECTOR)     ; GO TO REST OF IRQ

;
;SUBROUTINES
;
HIDIGT
   LSR A
   LSR A            ; MOVE HIGH NYBBLE INTO LOW
   LSR A
   LSR A
   CLC
   ADC #DIGIT       ; ADD $30 FOR SCREEN CODE
   RTS

LODIGT
   AND #TRNCLO      ; MASK OFF HIGH NYBBLE
   CLC
   ADC #DIGIT       ; ADD $30 FOR SCREEN CODE
   RTS

;
;CONVERT BINARY TO BCD
;
BINBCD
   LDY #$FF
   SEC
D10
   INY
   SBC #10          ; SUBTRACT 10 UNTIL -VE
   BCS D10
   ADC #10          ; ADD 10 BACK ON
   STA TEMP1        ; STORE REMAINDER
   TYA              ; GET NUMBER OF 10S SUBTRACTED
   ASL A
   ASL A
   ASL A            ; SHIFT INTO HIGH NYBBLE
   ASL A
   ORA TEMP1        ; PUT REMAINDER IN LOW NYBBLE
   RTS
```

Nachdruck mit freundlicher Genehmigung der Autoren und Ellis Horwood Ltd. Aus: ‚Mastering The Commodore 64' von Jones & Carpenter.

Tor zur Außenwelt

In dieser zweiteiligen Artikelfolge untersuchen wir das Ein- und Ausgabesystem des Commodore 64. Wir sehen uns dabei zunächst die Peripherieschnittstellen des Computers an.

Unser umseitiges Photo zeigt die Ausgänge des Commodore 64. Neben den Buchsen für Cassette, Audio/Video und TV stehen dem Programmierer drei E/A-Schnittstellen zur Verfügung. Von links nach rechts sind das die Erweiterungsschnittstelle (auch Cartridge-Port genannt), der serielle Ausgang (serieller Bus) und der User Port.

● **Der serielle Ausgang:** Über diese sechspolige DIN-Buchse werden serielle Commodoredrucker und das Diskettenlaufwerk 1541 angeschlossen. Bei jeder Gerätenummer (Device-Number) außer 2 bezieht sich der OPEN-Befehl immer auf diesen Ausgang. OPEN2,8,2,„DATEINAME" spricht beispielsweise das Diskettenlaufwerk über den seriellen Ausgang an und eröffnet dort eine Datei. Die serielle Schnittstelle sollte möglichst nur mit Commodoregeräten und auch nur über BASIC oder die E/A-Routinen eingesetzt werden.

Es gibt jedoch auch Schnittstellenkarten, die die seriellen Signale dieses Ausgangs in parallele IEEE-Daten umwandeln. Damit lassen sich etwa Peripheriegeräte des PET (beispielsweise die Diskettenstation 4040) bei entsprechendem Bedarf einsetzen.

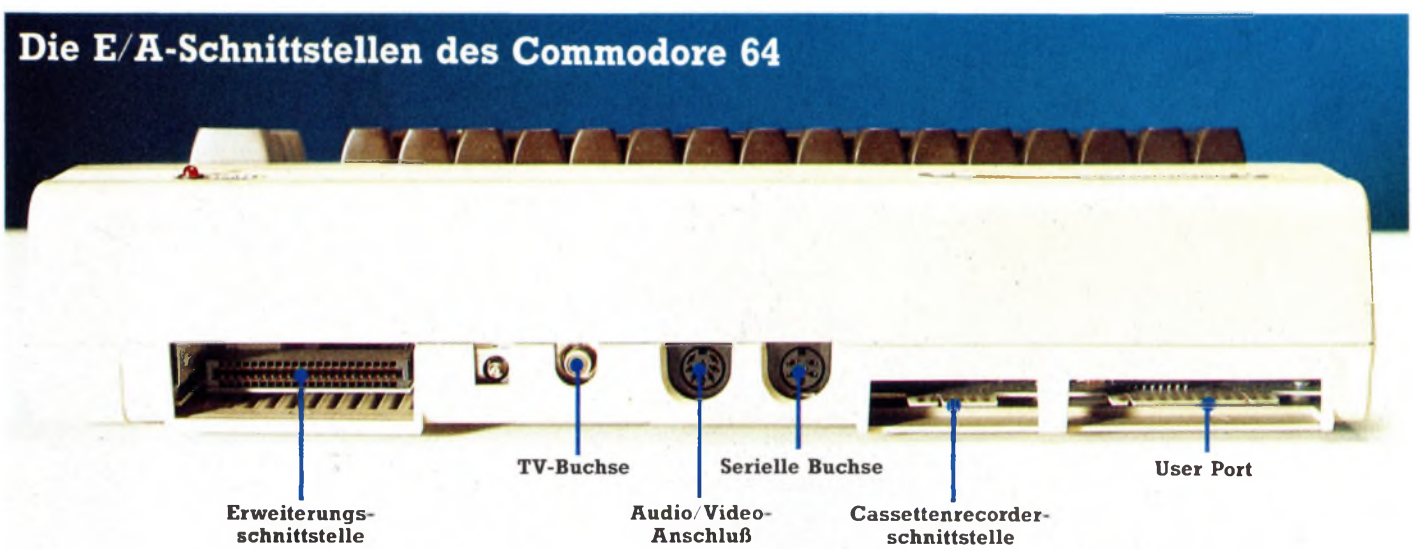
● **Der User Port:** Diese flache 24polige Steckleiste kann für serielle wie auch für parallele Datenübertragung eingesetzt werden. Zum Beispiel läßt sich darüber ein Drucker außerhalb des Commodorestandards an den C 64 anschließen. Der Drucker wird dabei als RS232-Gerät gesteuert.

Die Anwenderschnittstelle kann auch für die parallele Acht-Bit-Datenübertragung eingesetzt werden. Da das OS keine Treiberroutinen für die parallele Datenübertragung besitzt, müssen Sie sich jedoch Ihre eigenen Treiber (Maschinencodeprogramme, die Peripheriegeräte steuern) schreiben. In der nächsten Folge stellen wir ein Programm namens „Parawedge" – ein interruptgesteuertes, paralleles Datenübertragungsprogramm – vor. Das Programm kann für die Übermittlung eines Speicherblockes eingesetzt werden (z. B. ein BASIC-Programm von einem Commodore 64 zum anderen).

● **Die Erweiterungssteckleiste:** Diese 44-polige Steckleiste bietet Zugang zu allen Systembussen und den Daten- und Adreßleitungen des Commodore 64. Außer Spielcartridges werden hier auch die parallelen IEEE-Cartridges angeschlossen, mit denen der C 64 Peripheriegeräte des PET betreiben kann. Über diese Steckleiste werden auch Geräte oder Software angeschlossen, die die Steuerung der Maschine fast vollständig übernehmen.

E/A-Befehle

Die OS-Routinen für Ein- und Ausgabe (im Englischen „Kernel" genannt) sind Maschinencodeprogramme, die von E/A-Befehlen des BASIC (OPEN, CLOSE, GET#, PRINT# etc.) aufgerufen werden. Auf dem Commodore sind diese Routinen dem Maschinencodeprogram-



mierer leicht zugänglich. Ein kurzes Maschinencodeprogramm wird Ihnen zeigen, wie LOAD gesteuert wird.

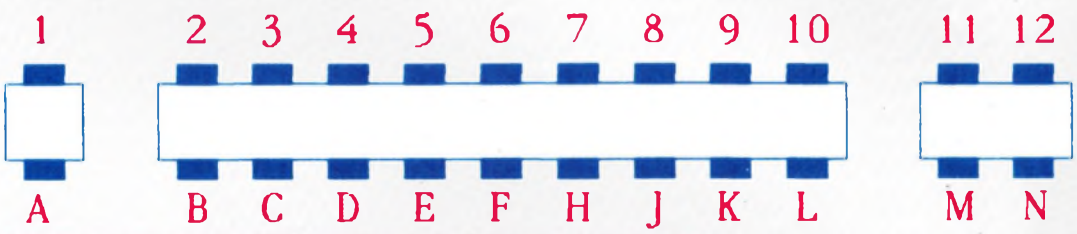
Wir haben weiterhin untersucht, welche Möglichkeiten dem C 64 über die RS232-Schnittstelle zur Verfügung stehen, und wie damit ein Modem betrieben wird.

Auf die Schnelle

Die Systemroutinen lassen sich jedoch nicht für alle E/A-Vorgänge einsetzen. Für Geräte, die nahe beeinander stehen, benötigt man Parallelverbindungen mit hohen Übertragsgeschwindigkeiten statt der seriellen Kommunikation, die eher für weiter entfernte Geräte geeignet ist. Da der C 64 keine Systemroutinen für die Steuerung des User Ports besitzt, müssen Sie die beiden 6526 CIAs (Complex Interface Adaptors) selbst programmieren. In der nächsten Folge zeigen wir anhand einer Assemblerroutine für den Acht-Bit parallelen Datentransfer, wie Programme dieser Art aussehen. Die Routine ermöglicht es dem Computer, über den User Port Daten zu lesen oder zu senden, während gleichzeitig ein BASIC-Programm abläuft. Dieser Ablauf wird durch die Interruptsteuerung des Schreib-/Lesecodes möglich.

Ein Beispiel für die IRQ-Anforderung – den „Programmkeil" – haben wir schon früher gezeigt. Hier arbeiten wir mit NMIs, da die Flagleitung (die zweite Interruptleitung des 6510) des User Ports zur Erzeugung eines (nichtmaskierbaren) NMI-Interrupts eingesetzt werden kann.

User Port



Oben

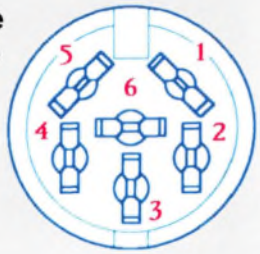
Kontakt	Typ	Allgemeine Bedeutung
1	GND	Erde
2	-	+5 Volt mit 100 mA Maximum
3	-	System RESET
4	CNT 1	CIA # 1 serieller Ausgang – Zähler
5	SP 1	CIA # 1 serieller Ausgang
6	CNT 2	CIA # 2 serieller Ausgang – Zähler
7	SP 2	CIA # 2 serieller Ausgang
8	PC 2	Handshakeleitung von CIA #2
9	-	Diese Leitung ist mit ATN des seriellen Ausgangs verbunden
10	-	9V AC +Phase
11	-	9V AC –Phase
12	-	Erde

Anmerkungen:
1) 3-Leitungen = Xon/Xoff Protokoll
X-Leitungen = CTS/RTS Protokoll
2) (*) Diese Leitungen können nicht nur über die RS232 E/A-Routinen des OS adressiert werden, sondern lassen sich auch direkt dem User Port zuordnen. Sie werden für die Steuerung von externen Geräten, wie Bufferboxen oder dem Bodenroboter unserer Bastelserie, eingesetzt.

Unten

Kanal	Typ	RS232 Funktion (*)	Eingesetzt in 3-/x-Ltg.
A	GND	Erde	Beide
B	FLAG2	Empfangsdaten -ein	Beide
C	PB0	Empfangsdaten -ein	Beide
D	PB1	Sendeteil schalten (RTS)-aus	Beide (hoch in 3–1)
E	PB2	Sendegerät bereit (DTR)-aus	Beide (hoch in 3–1)
F	PB3	akustisches Signal (RI)-ein	–
H	PB4	Signal empfangen (DCD)-ein	Nur X-Leitung
J	PB5	Nicht eingesetzt	–
K	PB6	sendebereit (CTS)-ein	Nur X-Leitung
L	PB7	Empfangsgerät bereit (DSR) -ein	Nur X-Leitung
M	PA2	Sendedaten -aus	Beide
N	GND	Masse	Beide

Serielle Buchse



Kanal	Typ	Bemerkung
1	SERIAL SRQIN	Sendeanforderung
2	GND	
3	SERIAL ATN I/O	"Attention"-Signal für Geräte am Port
4	SERIAL CLK I/O	Taktgeber für seriellen Ausgang
5	SERIAL DATA I/O	Übertragungsltg. für ein Datenbit
6	RESET	Leitung für Hardware-RESET

Erweiterungs-schnittstelle

Oben

Kanal	Typ	Bemerkung
1	GND	
2	+5v	
3	+5v	
4	IRQ	Interruptanforderung
5	R/W	Lesen/Schreiben EIN
6	Dot clock	
7	I/O 1	
8	GAME	
9	EXROM	
10	I/O 2	
11	ROML	
12	BA	
13	DMA	
14	D7	
15	D6	
16	D5	
17	D4	8 Datenleitungen
18	D3	
19	D2	
20	D1	
21	D0	
22	GND	

Unten

Kanal	Typ	Bemerkung
A	GND	
B	ROMH	
C	RESET	Hardware-RESET
D	NMI	Nicht-mask. Interrupt
E	S 02	
F	A15	
H	A14	
J	A13	
K	A12	
L	A11	
M	A10	
N	A9	
P	A8	16 Adreßleitungen
R	A7	
S	A6	
T	A5	
U	A4	
V	A3	
W	A2	
X	A1	
Y	A0	
Z	GND	

Die E/A-Routinen in Aktion

Der Ein-/Ausgabeteil des OS besteht aus einer Reihe von Subroutinen, die vom BASIC oder vom Maschinencode aus aufgerufen werden können. Das E/A-ROM reicht von $E000 bis $FFFF, doch werden die eigentlichen Routinen über die Pointer einer „Sprungtabelle" aufgerufen, die sich an der Obergrenze des Speichers befindet. Durch die Sprungtabelle läßt sich der Code auch auf anderen Commodoregeräten einsetzen, da sich die Adresse der Tabelle nie ändert – auch wenn die E/A-Routinen umgeschrieben werden.

Um E/A-Routinen einsetzen zu können, brauchen Sie weitere Informationen, da die Routinen nur dann funktionieren, wenn alle Mechanismen berücksichtigt sind.

Der folgende Code ist ein typisches Beispiel für den Einsatz der E/A-Routine LOAD bei der Ausführung eines verschiebbaren LOAD. Speichern Sie zunächst die ASC-Codes (Commodores ASCII-Version) des Dateinamens in aufeinanderfolgende Adressen und rufen Sie dann diesen Code auf:

```
LDA #$01      Logische Dateinummer (lfn)
LDX #$08      Devicenummer (Diskette)
LDY #$00      Sekundäradresse (#$00 ergibt
              verschiebbares Laden)
JSR $FFBA     E/A-Routine SLFS (lfn und Sekun-
              däradresse anlegen)
LDA #$0A      Länge Dateiname (z. B. dez. 10)
LDX PLO       Lo-byte Zeiger auf die Startadresse
              des Dateinamens
LDY PHI       Hi-byte Zeiger auf die Startadresse
              des Dateinamens
JSR #FFBD     E/A-Routine SETNAM (Dateiname
              anlegen)
LDA #$00      Laden = #$00 / Verif. = #$01
LDX DLO       Zieladresse (Anfang) – lo-byte
LDY DHI       Zieladresse (Anfang) – hi-byte
JSR $FFD5     E/A-Routine LOAD
```

Der Einsatz der RS232-Schnittstelle

Beim Einsatz von OS-Routinen zur Steuerung der RS232-Schnittstelle sollten Sie folgende Punkte beachten:

● Der Commodore 64 arbeitet mit Spannungen von 0V bis 5V, der RS232-Standard jedoch mit -12V bis 12V. Außer bei der Kommunikation mit einem Commodore 64 muß die Spannung daher umgewandelt werden. Commodore bietet für diese Aufgabe eine RS232-Cartridge an.

● Da die ASC-Codes des Commodore sich von denen des ASCII-Standards unterscheiden, benötigen Sie für das Senden und Empfangen je ein Umwandlungsarray.

● Bei jedem Eröffnen (OPEN) eines RS232-Kanals löscht das OS mit CLR automatisch die eingesetzten Register. Dabei gehen die Variablenwerte des laufenden BASIC-Programms verloren, so daß GOSUB-Befehle Fehlermeldungen auslösen, wenn sie auf RETURN treffen. Der Löschvorgang wird von den E/A-Routinen ausgelöst, die an der Speicherobergrenze zwei Buffer zu je 256 Bytes reservieren.

● Bei langen BASIC-Programmen und/oder bei einer großen Anzahl von Stringzuordnungen muß früher oder später der Speicher bereinigt werden. Dabei können eingehende Daten verlorengehen.

Der Befehl für die Eröffnung eines RS232-Kanals von BASIC aus lautet:

```
OPEN2,2,3,CHR$(CTRL)+CHR$(COMM)
```

CTRL und COMM sind die „Control-" und „Command"-Bytes, die die Informationen zur Kanaleröffnung enthalten. Beachten Sie, daß CTRL und COMM keine Variablen, sondern zwei echte Bytes (oder PEEKs von zwei zuvor angelegten Speicherstellen) sein müssen, da jedes Bit der CTRL- und COMM-Bytes eine bestimmte Funktion hat (siehe Tabellen).

Beispielsweise wird die Eröffnung eines RS232-Kanals mit einem Stop-Bit, einem 7-Bit-Wort, 300 Baud (CTRL Byte gesamt = 38), gerader Parität, Voll-Duplex und drei Steuerleitungen (COMM Byte gesamt = 96) mit folgendem Befehl durchgeführt:

```
OPEN2,2,3,CHR$(38)+CHR$(96)
```

Bei der Festlegung der Baudrate müssen folgende Faktoren berücksichtigt werden: Für das Senden von Daten ist die Baudrate nicht kritisch, da die Geräte normalerweise geringe Geschwindigkeitsschwankungen tolerieren. Unter BASIC können Zeichen durchaus mit 2400 Baud über die RS232-Leitung gesandt werden. Dabei beträgt die Übertragungsrate der Zeichen 2400 Baud, doch sind die Pausen zwischen den einzelnen Bytes oft weitaus länger, so daß die tatsächliche Geschwindigkeit niedriger ist.

Beim Empfang von Daten ist die Situation jedoch grundlegend anders. In diesem Fall hat ein BASIC-Programm selbst bei 300 Baud kaum Zeit, sich ein Byte aus dem Eingabebuffer zu nehmen und zum Bildschirm zu schicken. Für den Empfang von Daten müssen Sie daher den Datenfluß „steuern".

1) Einlesen einer geringen Anzahl Bytes (je höher die Baudrate, desto kleiner die Zahl) mit GET#2,A$. Bearbeiten Sie diese Bytes schnell, oder speichern Sie sie zur späteren Verarbeitung in einem Array.

2) Stoppen des anderen Gerätes mit dem Befehl PRINT#2,CHR$(17).

3) Einlesen von weiteren Bytes, bis der eigentliche RS232-Buffer leer ist. Ausführliche Bearbeitung aller empfangenen Bytes, während das System prüft, ob Zeicheneingaben oder EOF-Signale, die etwa das Dateiende anzeigen, empfangen wurden.

4) Anforderung weiterer Daten von dem Sendegerät mit PRINT#2,CHR$(19) und zurück zu Schritt eins.

Ist ein RS232-Kanal einmal eröffnet, werden Bytes auf die übliche Weise mit PRINT# oder GET# (nicht aber INPUT#) gesendet oder empfangen. Sie sollten auch das Statusbyte ST testen, das für alle E/A-Programme mit ST=0 oder ST=8 anzeigt, daß das System fehlerfrei arbeitet.

CTRL-Byte

Bit 7	6	5	4	3	2	1	0	Funktion
—	—	—	X	0	0	0	1	50 Baud
—	—	—	X	0	0	1	0	75 Baud
—	—	—	X	0	0	1	1	110 Baud
—	—	—	X	0	1	0	0	134,5 Baud
—	—	—	X	0	1	0	1	150 Baud
—	—	—	X	0	1	1	0	300 Baud
—	—	—	X	0	1	1	1	1200 Baud
—	—	—	X	1	0	0	0	1800 Baud
—	—	—	X	1	0	1	0	2400 Baud
—	0	0	X	—	—	—	—	8-Bit data
—	0	1	X	—	—	—	—	7-Bit data
—	1	0	X	—	—	—	—	6-Bit data
—	1	1	X	—	—	—	—	5-Bit data
0	—	—	X	—	—	—	—	1 Stop-Bit
1	—	—	X	—	—	—	—	2 Stop-Bits

X = Ist egal

COMM-Byte

Bit 7	6	5	4	3	2	1	0	Funktion
—	—	—	—	X	X	X	0	Protokoll mit 3 Leitungen
—	—	—	—	X	X	X	1	Protokoll mit X Leitungen
—	—	—	0	X	X	X	—	Vollduplex
—	—	—	1	X	X	X	—	Halbduplex
—	—	0	—	X	X	X	—	Parity AUS
0	0	1	—	X	X	X	—	ungerade Parität
0	1	1	—	X	X	X	—	gerade Parität
1	0	1	—	X	X	X	—	Mark send p-check AUS
1	1	1	—	X	X	X	—	Space send p-check AUS

Kontakt zum CIA

In der vorigen Folge wurden die E/A-Möglichkeiten des C 64 behandelt. Wir wollen nun untersuchen, wie die beiden 6526 CIA Chips die Ein- und Ausgabe durchführen.

Der Commodore 64 besitzt zwei 6526 CIA- (Complex Interface Adaptor) Chips, die sich speziell um die Kommunikation mit der Außenwelt kümmern. Die CIAs sind jedoch nicht die einzigen Chips mit E/A-Funktionen – auch der 6510 und die Video Chips steuern Teilbereiche der Ein- und Ausgabe. Ein 6526-Chip besitzt zwei Acht-Bit-Datenausgänge, die beide über individuell programmierbare Leitungen verfügen. Der Chip kann Daten im Acht- oder 16-Bit-Format übertragen und hat außerdem zwei 16-Bit-Timer, die sich verbinden lassen. Er verfügt weiterhin über ein Acht-Bit-Shift-Register für die serielle Datenübertragung und eine programmierbare 24-Stunden-Uhr für vorhersehbare Ereignisse.

„Data Ready"

Der Chip hat zwei Handshake-Leitungen: PC und Flag. PC wird für einen Taktzyklus auf niedrig geschaltet, nachdem Daten zum Ausgang B des 6526 gesandt wurden, und kann daher einem externen Gerät das Signal „Data Ready" anzeigen. Die Flag-Leitung kann die Steuersignale eines externen Gerätes entgegennehmen. Damit läßt sich beispielsweise das Flag-Bit des Interrupt-Registers setzen.

Auf dem Commodore 64 steuert jeder der beiden 6526 Chips unterschiedliche E/A-Bereiche: CIA #1 (mit Basisadresse bei $DC00) ist für Tastatur und Joystick abgestellt und CIA #2 (mit Basisadresse bei $DD00) für die Datensteuerung der seriellen Schnittstelle und des User Ports. Der Video Chip führt die E/A zum Monitor aus, während der 6510 den Recorderausgang direkt steuert.

Das Programm Parawedge zeigt, wie der 6526 direkt für E/A programmiert wird. Die Routine ist ein NMI-„Keil" und arbeitet mit der Flag-Leitung. Sie sendet einen bestimmten Speicherblock als parallele Daten zum User Port oder empfängt Acht-Bit parallele Daten, bis der definierte Speicherblock voll ist. Da die Daten per NMI ausgesandt oder empfangen werden, kann die Maschine in der übrigen Zeit andere Aufgaben ausführen. Bei hohen Übertragungsraten muß sich der Commodore 64 allerdings hauptsächlich den NMI-Routinen widmen und andere Vorgänge vorübergehend „vernachlässigen".

Parawedge richtet auf dem Commodore 64 eine Acht-Bit parallele Datenübertragung mit einem externen Gerät (beispielsweise einem anderen Computer oder einem Paralleldrukker) ein, die in beide Richtungen funktioniert und über den User Port abläuft. Dabei werden die Kontakte PB0 bis PB7 für die Datenübertragung genutzt, Flag 2 für eingehende Handshakesignale, PA2 für das Signal „sendebereit" und PC2 für die Bedingung „Daten korrekt empfangen". Vor Einsatz des Programms müssen Sie zunächst festlegen, welchen RAM-Bereich Sie für die ein- oder ausgehenden Daten einsetzen wollen. Dabei werden die Anfangs- und Endadresse (im Format Lo-/Hi-Byte) mit POKE an die vier Speicherstellen von 50768 bis 50771 übergeben.

Speicherstelle 50772 gibt an, ob das Programm senden oder empfangen soll. Eine Eins schaltet auf Sendung, eine Null auf Empfang. Der eigentliche Programmcode beginnt schließlich bei SYS 50775.

Das Programm Parawedge für den Commodore 64

BASIC-Ladeprogramm

```
1000 REM ** PARAWEDGE BASIC LOADER **
1010 DATA173,84,198,208,61,169,0,141,3
1020 DATA221,169,144,141,13,221,173,2
1030 DATA221,9,4,141,2,221,173,0,221,9
1040 DATA4,141,0,221,173,80,198,133,251
1050 DATA173,81,198,133,252,173,24,3
1060 DATA141,85,198,173,25,3,141,86,198
1070 DATA120,169,188,141,24,3,169,198
1080 DATA141,25,3,88,96,169,255,141,3
1090 DATA221,169,144,141,13,221,173,24
1100 DATA3,141,85,198,173,25,3,141,86
1110 DATA198,120,169,234,141,24,3,169
1120 DATA198,141,25,3,88,96,169,144,44
1130 DATA13,221,240,36,173,1,221,145
1140 DATA251,230,251,208,2,230,252,173
1150 DATA82,198,197,251,173,83,198,229
1160 DATA252,144,49,173,0,221,41,252
1170 DATA141,0,221,9,4,141,0,221,108,85
1180 DATA198,169,144,44,13,221,240,246
1190 DATA177,251,141,1,221,230,251,208
1200 DATA2,230,252,173,82,198,197,251
1210 DATA173,83,198,229,252,144,3,108
1220 DATA85,198,120,173,85,198,141,24,3
1230 DATA173,86,198,141,25,3,88,108,24
1240 DATA3
1250 DATA25596:REM*CHECKSUM*
1260 CC=0
1270 FORI=50775TO50971
1280 READX:CC=CC+X:POKEI,X
1290 NEXT
1300 READX:IFCC<>XTHENPRINT"CHECKSUM
  ERROR"
1310 END
```

Das Assemblerlisting von Parawedge kann eingegeben und mit einem Assembler in den Maschinencode übersetzt werden. Über das BASIC-Ladeprogramm läßt sich das Programm auch mit DATA-Befehlen einlesen.

Das Programm im Assemblercode

```
;**************************************
;*                                    *
;*       PARAWEDGE - A SEND/RECEIVE   *
;*                                    *
;*  WEDGE PROGRAM FOR 8 BIT PARALLEL  *
;*                                    *
;*     COMMUNICATIONS ON THE CBM 64   *
;*                                    *
;**************************************
;
  CIA2 = $DD00      ;6526 CHIP BASE ADDR
  OUTPUT = $FF
  INPUT = $00
  OUTSHK = $04
  INTMSK = $90
  TOGHI = $04
  TOGLO = $FC
  NMIVEC = $0318
  ZPTEMP = $FB
  * = $C650
  START *=*+2       ;START ADDRESS
  END *=*+2         ;END ADDRESS
  MODE *=*+1        ;INPUT/OUTPUT FLAG
  VECTOR *=*+2      ;STORAGE FOR NMI VECTOR

  LDA MODE          ;INPUT OR OUTPUT
  BNE OUTDAT        ;BRANCH IF OUTPUT
  LDA #INPUT
  STA CIA2+3        ;SET DDR FOR INPUT
  LDA #INTMSK
  STA CIA2+13       ;FLAG INTERRUPTS DISABLED
  LDA CIA2+2
  ORA #OUTSHK
  STA CIA2+2        ;SET PA2 FOR OUTPUT
  LDA CIA2
  ORA #TOGHI
  STA CIA2          ;SET HANDSHAKE LINE PA2 HIGH
  LDA START
  STA ZPTEMP
  LDA START+1       ;MOVE POINTERS TO ZERO PAGE 0
  STA ZPTEMP+1
 ;
 ;INITIALISE INPUT WEDGE
 ;
  LDA NMIVEC
  STA VECTOR        ;SAVE OLD NMI VECTOR
  LDA NMIVEC+1
  STA VECTOR+1
  SEI               ;
  LDA #<NXTIN       ;
  STA NMIVEC        ;INSERT DATA-INPUT WEDGE
  LDA #>NXTIN       ;
  STA NMIVEC+1      ;
  CLI               ;
  RTS
 ;
 ;INITIALISE OUTPUT WEDGE
 ;
OUTDAT
  LDA #OUTPUT
  STA CIA2+3        ;SET DDR FOR OUTPUT
  LDA #INTMSK
  STA CIA2+13       ;FLAG INTS DISABLED
 ;
 ;
  LDA NMIVEC
  STA VECTOR
  LDA NMIVEC+1      ;SAVE OLD NMI VECTOR
  STA VECTOR+1
  SEI               ;
  LDA #<NXTOUT      ;
  STA NMIVEC        ;INSERT DATA-OUTPUT WEDGE
  LDA #>NXTOUT      ;
  STA NMIVEC+1      ;
  CLI               ;
  RTS
 ;
 ;
 ;INPUT DATA SERVICE ROUTINE
 ;
 ;
NXTIN
  LDA #INTMSK       ;CHECK ICR
  BIT CIA2+13       ;INT CAUSED BY FLAG?
  BEQ NOTCOM        ;NO.. NORMAL NMI
 ;
 ;OK BYTE ON PORT
 ;
  LDA CIA2+1        ;READ BYTE
  STA (ZPTEMP),Y    ;STORE IN MEMORY
  INC ZPTEMP
  BNE TEST1         ;INCREMENT POINTER
  INC ZPTEMP+1
 ;
TEST1
  LDA END
  CMP ZPTEMP
  LDA END+1         ;CHECK TO SEE IF ENDED
  SBC ZPTEMP+1
  BCC DONE          ;BRANCH IF FINISHED
 ;
 ;TELL DEVICE READY FOR NEXT BYTE
 ;
  LDA CIA2
  AND #TOGLO
  STA CIA2          ;TOGGLE PA2 LOW THEN HIGH
  ORA #TOGHI
  STA CIA2
 ;
 ;NOW DO NORMAL NMI ROUTINE
 ;

NOTCOM
  JMP (VECTOR)


 ;
 ;
 ;OUTPUT DATA SERVICE ROUTINE
 ;
 ;
NXTOUT
  LDA #INTMSK       ;CHECK ICR
  BIT CIA2+13       ;INTERRUPT CAUSED BY FLAG?
  BEQ NOTCOM        ;NO.. DO NORMAL NMI
 ;
 ;OK SEND BYTE
 ;
  LDA (ZPTEMP),Y    ;GET BYTE FROM MEMORY
  STA CIA2+1        ;OUTPUT IT. PC WILL GO
                    ;LOW FOR 1 CYCLE
  INC ZPTEMP
  BNE TEST2         ;INCREMENT POINTER
  INC ZPTEMP+1

TEST2
  LDA END
  CMP ZPTEMP
  LDA END+1         ;CHECK TO SEE IF ENDED
  SBC ZPTEMP+1
  BCC DONE          ;BRANCH IF DONE
 ;
 ;CONTINUE NORMAL NMI ROUTINE
 ;
  JMP (VECTOR)
 ;
 ;
 ;FINISHED REMOVE WEDGE
 ;
 ;
DONE
  SEI
  LDA VECTOR
  STA NMIVEC        ;RESET NMI VECTOR TO
  LDA VECTOR+1      ;ORIGINAL VALUE
  STA NMIVEC+1
  CLI
  JMP (NMIVEC)
```

Parawedge stammt aus ‚Mastering The Commodore 64' von Jones und Carpenter und wird mit freundlicher Genehmigung der Autoren und der Ellis Horwood Ltd. abgedruckt.

Farbenfreude

Wir untersuchen, wie über die Register des VIC-II des Commodore 64 unterschiedliche Darstellungsmodi erzeugt werden. Wir generieren Farbdarstellungen und zeigen, wie der VIC-II den Speicher anspricht.

Auf dem Commodore 64 gibt es acht grundlegende Grafikmodi. In niedriger Auflösung befindet sich der Zeichensatz im RAM oder ROM und kann auf drei verschiedene Weisen angezeigt werden: Standardmodus, Vielfarbmodus und erweiterter Farbmodus. Weiterhin gibt es zwei Arten von hoher Auflösung: Standardmodus und Vielfarbmodus. Außer den Grundarten sind weitere Variationen möglich: Der Bildschirm kann auf 38 Spalten (statt der normalen 40) gesetzt werden und/oder auf 24 Zeilen (Standard sind 25). Die letzten beiden Darstellungsarten werden normalerweise mit dem „sanften" horizontalen und vertikalen Scrollen des Bildschirms eingesetzt, das auf dem Commodore 64 am besten vom Maschinencode aus gesteuert wird.

Wenn ein langes Programm mit hochauflösender Grafik arbeitet, wird leicht der Speicherplatz knapp. Der Bildschirmspeicher für niedrige und hohe Auflösung läßt sich jedoch an fast jede Position des Arbeitsspeichers verlegen. Wir sehen uns daher zunächst an, wie eine Verlegung durchgeführt wird. Wenn Ihr Programm ausschließlich Maschinencode enthält, können Sie den Bildschirmspeicher für hohe Auflösung sogar „hinter" das ROM des BASIC-Interpreters legen und so die acht KByte des Interpreter-ROMs besser einsetzen. Interessanterweise benötigt die hochauflösende Grafik gerade acht KByte des vorhandenen Speicherplatzes.

Auf Empfang

Der 6555/67 Video-Interface-Chip (VIC-II) hat die Aufgabe, Bildschirmdaten zu erzeugen, die dann an den Fernseher oder Monitor weitergeleitet werden. Der VIC-II muß dazu Daten aus dem RAM oder ROM lesen. Um das recht komplizierte Verhalten des C64 besser verstehen zu können, untersuchen wir zunächst, wie der VIC-II Daten erhält.

Im Vielfarbmodus kann ein Zeichenfeld vier statt zwei Farben haben. Damit das möglich ist, besteht die horizontale Auflösung hier nicht mehr aus einzelnen Pixeln, sondern aus Pixelpaaren. Der Vielfarbmodus läßt sich im hohen oder niedrigen Grafikmodus einsetzen, wenn auch die Pixelfarben bei der hohen Auflösung auf andere Weise bestimmt werden. Die folgenden BASIC-POKEs schalten den Vielfarbmodus an und ab:

```
POKE 53270,PEEK(53270)OR16
POKE 53270,PEEK(53270)AND239
```

Wenn bei niedriger Auflösung und Vielfarbmodus das Bit 4 des entsprechenden Farbnybbles (ein Nybble ist ein Halbbyte oder vier Bits) auf Eins steht, wird das Zeichen im Vielfarbmodus interpretiert, wobei die niederwertigen drei Bits die Farbe festlegen. Zeichen mit Farbnybbles im Bereich von 0 bis 7 werden daher normal interpretiert, Zeichen mit Farbcode zwischen 8 und 15 hingegen im Vielfarbmodus. Tabelle 1 zeigt, wie die Farben der Pixelpaare festgelegt werden.

Erweiterter Farbmodus

Durch Austauschen der Adressen 53282 und 53282 können Sie im Handumdrehen die Farbe jedes Pixelpaares wechseln. Der Vielfarbmodus eignet sich besonders für anwenderdefinierte Zeichen, da hier beim Bitmuster die Pixelpaare berücksichtigt wurden.

Der „erweiterte Farbmodus" bietet zusätzliche Darstellungsmöglichkeiten. Diese Farbart steuert die Hintergrundfarbe der ersten 64 Zeichen der Zeichenmatrix. Der erweiterte Farbmodus läßt sich jedoch nicht zusammen mit dem Vielfarbmodus einsetzen. Je eine BASIC-Zeile schaltet den erweiterten Farbmodus an bzw. ab:

```
POKE 53265,PEEK(53265)OR64
POKE 53265,PEEK(53265)AND191
```

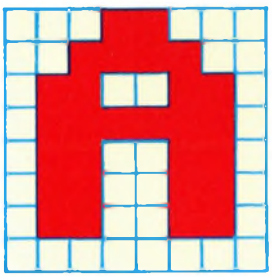
Auf dem Bildschirm kann nun das gleiche Zeichen in vier verschiedenen Hintergrundfarben erscheinen (eine davon ist die des Bildschirms). Die verbleibenden Zeichen lassen

Tabelle 1

Bitmuster des Pixelpaares		Farbe	Festgelegt durch
0	0	Bildschirmhintergrund	53281 ($D021) Bits 0 bis 4
0	1	Vielfarbmodus #1	53282 ($D022) Bits 0 bis 4
1	0	Vielfarbmodus #2	53283 ($D023) Bits 0 bis 4
1	1	Vordergrundfarbe	Bits 0 bis 3 des Farbnybbles

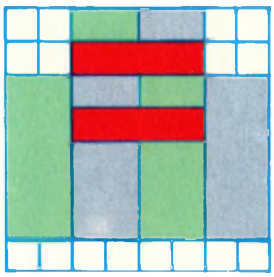
Tabelle 2

Bildschirmcode	Bit 7	Bit 6	Register für Hintergrundfarbe
64–127	0	1	53282 ($D022) Erweiterte Farbe 1
128–191	1	0	53283 ($D023) Erweiterte Farbe 2
192–255	1	1	53284 ($D024) Erweiterte Farbe 3



„A" im normalen Anzeigemodus

0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0



„A" im Vielfarbmodus

00	01	10	00
00	11	11	00
01	10	01	10
01	11	11	10
01	10	01	10
01	10	01	10
01	10	01	10
00	00	00	00

Farbschlüssel

01 Vielfarbmodus 1

10 Vielfarbmodus 2

11 Vordergrundfarbe

00 Hintergrundfarbe

Im normalen Anzeigemodus des C 64 wird jedes Bit eines Zeichens, das auf Eins steht, in Vordergrundfarbe dargestellt und jedes Bit, das auf Null steht, in Hintergrundfarbe. Beim Vielfarbmodus werden die Bitmuster jedoch als Bitpaare interpretiert. Die vier möglichen Kombinationen eines Bitpaares stellen außer den Farben des Vorder- und Hintergrundes noch zwei zusätzliche Farben dar.

sich im erweiterten Farbmodus nicht darstellen, da Bit 6 und 7 des Bildschirmcodes für die indirekte Steuerung der Zeichenfarbe gebraucht wird. So ist 1 beispielsweise der Bildschirmcode für „A" und 65 der für „A" in Negativdarstellung. Wenn Sie jedoch 65 in den Bildschirmspeicher POKEn, sehen Sie kein inverses „A", sondern ein normales „A", dessen Hintergrundfarbe der Inhalt der Adresse 53282 ($D022) bestimmt. Auf die gleiche Weise erzeugt ein POKE 129 in den Bildschirmspeicher ein normales „A", jedoch mit der Hintergrundfarbe, die die Adresse 53283 ($D023) angibt. Tabelle 2 zeigt die Beziehung zwischen Bildschirmcodes und Farbregistern.

Der VIC-II Chip arbeitet mit einem einfachen Speicherschema. Er kann jeweils nur einen von vier Blöcken zu je 16 KByte – Bank genannt – „sehen". Stellen Sie sich diese Banks am besten als Fenster des VIC-II zum Arbeitsspeicher vor. Die Basisadresse des Fensters ist softwaregesteuert und kann einen von vier Werten annehmen.

```
1000 REM**BANK FUER DAS VIC FENSTER
     WAEHLEN**
1010 POKE 56578,PEEK(56578)OR3:REM
     DDR
     CIA#2 BITS 0,1 ZUR AUSGABE
1020 WD=3:REN WAEHLT NORMALES
     FENSTER
1030 POKE 56576,(PEEK(56576)AND252)OR
     WD:
     REM PORT A CIA#1 BITS 0,1
```

WD kann Werte von 0 bis 3 annehmen. Der aktuelle Wert von WD läßt sich jederzeit mit PEEK(56576)AND3 abfragen. Die Adresse der Untergrenze des Speicherfensters wird mit der Formel WB=16384*(3-WD) errechnet, wobei WD die entsprechenden Adressen liefert:

WD	Fenster Startadresse
0	49152 ($C000)
1	32768 ($8000)
2	16384 ($4000)
3	0 ($0000)

Im Inneren des VIC-II Fensters erwartet der 6510 den Bildschirmspeicher und ein Abbild des Zeichen-ROMs in niedriger Auflösung (oder in hoher, wenn dieser Grafikmodus angeschaltet ist). Die CPU findet dort auch ihre Sprite-Daten. Die Pointer auf die acht Sprites sind am Ende des Bildschirmspeichers untergebracht und müssen bei einer Verlegung ebenfalls verschoben werden.

Die vier höherwertigen Bits des VIC-II Steuerregisters bei der Adresse 53272 ($D018) geben an, welches Offset auf die Basis des aktuellen VIC-II Fensters den Anfang des Bildschirmspeichers angibt. Mit diesen vier Bits kann der Bildschirmspeicher in jeden der 16K-Fensterblöcke gelegt werden.

```
1040 REM**OFFSET DES BILDSCHIRMS VON
     DER BASIS DES FENSTERS
     WAEHLEN**
1050 SO=1:REM LEGT NORMALES OFFSET
     FEST
1060 POKE 53272,(PEEK(53272)AND15)
     OR16*SO
```

SO kann hier Werte von 0 bis 15 annehmen. Der aktuelle Wert von SO kann jederzeit mit PEEK(53272)AND15 festgestellt werden. Die entsprechende Basisadresse des aktuellen Bildschirmspeichers wird entweder mit SC=WB+1024*SO (Fensterbasis plus Offset) errechnet oder mit:

```
SC=16384*(3-(PEEK(56576)AND3))+64*(PEEK
53272)AND240)
```

Da das Farb-RAM sich bei der Verlegung des Bildschirmspeichers nicht mitbewegt, gibt es eine Schwierigkeit. Für einen kompletten zweiten Bildschirmspeicher brauchen Sie eine kurze Maschinencoderoutine, die den Farbspeicher in und aus einem Buffer bewegt.

Zeigereinstellung

Wenn Sie den neuen Bildschirm mit PRINT ansprechen wollen, müssen Sie weiterhin dem Betriebssystem mitteilen, wo sich der neue Bildschirmspeicher befindet. Dies kann mit einem POKE (oder STA) auf Adresse 648 ($0288) geschehen, da dort der Pointer auf das höherwertige Byte der Bildschirmspeicher-Basisadresse zeigt. Wenn SC wie beschrieben berechnet wurde, führt

```
POKE 648,INT(SC/256)
```

diese Aufgabe aus. Die Basisadresse der Zeichenmatrix oder des hochauflösenden Bildschirmes wird folgendermaßen festgelegt:

```
1070 REM** HOCHAUFLOESENDES/CM
     OFFSET FUER DIE FENSTERBASIS
     WAEHLEN**
1080 HO=4:REM NORMALES OFFSET
     FESTLEGEN
1090 POKE 53272,(PEEK(53272)AND240)OR
     2*HO
```

Prinzipiell kann HO jeden Wert von 0 bis 7 annehmen; in der Praxis sind die Möglichkeiten jedoch eingeschränkt. Wenn WD 1 oder 3 entspricht, darf HO nicht die Werte 2 oder 3 annehmen, da der VIC-II dort das normale ROM-Zeichenabbild erwartet. Zu hohe Werte von HO verlegen die Obergrenze des hochauflösenden Speichers außerhalb des Bereichs, den der VIC-II ansprechen kann. Die entsprechende Basisadresse der aktuellen Zeichenmatrix (bzw. des hochauflösenden Bildschirmspeichers) wird entweder mit CM=WB+2048*HO (Fensterbasis plus Offset) errechnet, oder mit

```
CM=16384*(3-(PEEK(56576)AND3))+1024*
PEEK(53272)AND14)
```

Durch Setzen dieser Register lassen sich die Bildschirmspeicher in Bereiche verlegen, die sich für Ihr Programm am besten eignen.

Geborgte Zeit

Im zweiten Teil über die Grafiktalente des Commodore 64 sehen wir, wie der VIC-II Chip für seine Bildschirmanzeige Zeit vom 6510 „borgt". Das Beispielprogramm „Splitscreen" teilt den Bildschirm und aktiviert zwei unterschiedliche Grafikmodi gleichzeitig.

Um Bildschirmdaten überhaupt anzeigen zu können, muß der VIC-II Informationen aus dem Speicher holen. Ihm stehen dazu einige Adreßleitungen und alle Leitungen des Datenbusses zur Verfügung. Den Zugang zu ROM und RAM (die er sich mit dem 6510 teilt) verschafft sich der VIC-II mit der Technik des „direkten Speicherzugriffs" (DMA – direct memory access). Im Idealfall würde nun der VIC-II die Adreß- und Datenleitungen nur nutzen, wenn der 6510 sie nicht belegt. Die Abläufe der beiden Prozessoren wären dann „transparent". Doch leider ist der 6510 ein recht einfacher Prozessor mit nur wenigen Registern.

Der VIC-II muß aber, besonders bei mehreren Sprites, viele Daten lesen und hat nicht genug Zeit für eine transparente Arbeitsweise. Mit einer speziellen Steuerleitung (BA – Bus available) sendet er dem 6510 ein Wartesignal und verschafft sich so ausreichend Zeit zum Lesen der Bildschirmdaten.

Wenn der Video-Chip Zeit von dem 6510 benötigt, setzt er die BA-Leitung auf „low". Der 6510 kann dann gerade noch den aktuellen Befehl beenden, bevor der VIC-II die AEC-Leitung (Adress Enable Control) herunterschaltet und die Bustreiber des 6510 deaktiviert. Dieser Vorgang kommt einer „Betäubung" des 6510

„Splitscreen" für den Commodore 64

Ladeprogramm in BASIC

```
900 REM**************************
902 REM**     SPLITSCREEN 64     **
903 REM**      BASIC LOADER      **
910 REM** NOTE-IN THIS VERSION **
920 REM** HIRES UNDER BASIC SO **
930 REM**    CAN'T USE PEEK IN  **
940 REM**     PLOT SUBROUTINE   **
950 REM**************************
960 :
1000 REM SPLIT SCREEN LOAD AND TEST**
1010 DATA76,50,192,49,234,173,25,208
1020 DATA240,37,169,255,160,21,141,25
1030 DATA208,173,0,221,73,2,141,0,221
1040 DATA173,17,208,73,32,141,17,208,41
1050 DATA32,240,4,169,121,160,255,141
1060 DATA18,208,140,24,208,108,3,192
1070 DATA173,14,220,41,254,141,14,220
1080 DATA173,20,3,141,3,192,173,21,3
1090 DATA141,4,192,169,5,141,20,3,169
1100 DATA192,141,21,3,173,17,208,41,95
1110 DATA141,17,208,169,0,141,18,208
1120 DATA169,255,141,25,208,169,1,141
1130 DATA26,208,165,1,41,254,133,1,169
1140 DATA0,133,247,169,160,133,248,160
1150 DATA0,162,28,169,0,145,247,200,208
1160 DATA251,230,248,202,208,246,169
1170 DATA188,133,248,160,0,162,4,169
1180 DATA183,145,247,200,208,251,230
1190 DATA248,202,208,246,165,1,9,1,133
1200 DATA1,96,255
1210 DATA20633:REM*CHECKSUM*
1220 CC=0:FORI=0TO160
1230 READX:POKE49152+I,X:REM INSERT CODE
1240 CC=CC+X:NEXT
1250 READX:IFX<>CCTHENPRINT"CHECKSUM ERROR":STOP
1260 REM***TEST SPLIT SCREEN***
1270 SYS49152:REM SPLIT SCREEN
1280 CM = 40960:REM START HIRES
1290 REM** PLOT VERT  AXES **
1300 X=160:FORY=0TO70
1310 GOSUB1400:REM COMPUTE ADDRESS
1320 POKEM+R,2↑(7-C):REM PLOT POINT
1330 NEXT
1340 REM** PLOT HORIZ AXES **
1350 Y=37:FORX=0TO319
1360 GOSUB1400:REM COMPUTE ADDRESS
1370 POKEM+R,255:REM PLOT SEGMENT
1380 NEXT
1390 END
1400 REM** COMPUTE S/R **
1410 U=INT(Y/8):V=INT(X/8)
1420 R=YAND7:C=XAND7
1430 M=CM + (40*U+V)*8
1440 RETURN
```

Assemblerprogramm

```
;++++++++++++++++++++++++++
;++++++++++++++++++++++++++
;++                      ++
;++   SPLITSCREEN 64     ++
;++                      ++
;++++++++++++++++++++++++++
;++++++++++++++++++++++++++

;
 *= $C000
 JMP START
;
 MEM1 = $F7
 MEM2 = $F9
 MEM3 = $FB
 SCN = $FD
 CINV = $314     ;IRQ VECTOR
 TEMPIRQ *=*+2
;
NEWIRQ =*        ;NEW IRQ WEDGE START
;
 LDA $D019       ;LOOK AT THE INTERRUPT STATUS
 BEQ NOTVIC      ;WAS THE IRQ FROM VIC?          DDD
 LDA #$FF
 LDY #$15
 STA $D019       ;CLEAR INT LATCH.
 LDA $DD00
 EOR #$02
 STA $DD00       ;CHANGE BANKS
 LDA $D011
 EOR #$20        ;CHANGE BIT MAP MODE BIT
 STA $D011
 AND #$20        ;TEST BMM BIT
 BEQ LSCN        ;LOWER PART OF SCREEN
 LDA #$79        ;SET RASTER TO LINE 22          DDE
 LDY #$FF
LSCN
 STA $D012       ;SET NEW RASTER COMPARE
 STY $D018
NOTVIC
 JMP (TEMPIRQ)
;
START =*         ;INITIALISE IRQ WEDGE
;
 LDA $DC0E
 AND #$FE        ;DISABLE SCAN TIMER A
 STA $DC0E       ;CIA #1
 LDA CINV
 STA TEMPIRQ     ;
 LDA CINV+1      ;CHANGE IRQ VECTOR
 STA TEMPIRQ+1   ;
 LDA #<NEWIRQ
 STA CINV
 LDA #>NEWIRQ
 STA CINV+1
 LDA $D011
 AND #$5F        ;RESET BMM MODE
 STA $D011
 LDA #$00
 STA $D012
 LDA #$FF
 STA $D019       ;RESET IRQ LATCHES
 LDA #$01
 STA $D01A       ;ENABLE RASTER COMPARE
 LDA $01
 AND #$FE
 STA $01         ;BANK OUT BASIC ROM
 LDA #$00
 STA MEM1        ;SET UP 0 PAGE PTRS TO POINT
 LDA #$A0        ;TO START OF HIRES AREA BEHIND
 STA MEM1+1      ;BASIC ROM
 LDY #$00
 LDX #$1C
 LDA #$00        ;CLEAR HIRES
 STA (MEM1),Y                                    DDD
 INY
 BNE DDD
 INC MEM1+1
 DEX
 BNE DDD
 LDA #$BC
 STA MEM1+1
 LDY #$00
 LDX #$04
 LDA #$B7        ;SET COLOUR
 STA (MEM1),Y                                    DDE
 INY
 BNE DDE
 INC MEM1+1
 DEX
 BNE DDE
 LDA $01
 ORA #$01
 STA $01         ;RESTORE BASIC ROM
 RTS             ;BACK TO BASIC
```

gleich, der überhaupt nicht merkt, daß er nicht mehr im Spiel ist.

Der VIC-II „stiehlt" dem 6510 eine ganze Menge Zeit. So müssen beispielsweise nach Darstellung jeder achten Rasterzeile die Pointeradressen der Zeichen (die aus acht Pixelzeilen bestehen) geholt werden. Da in einer Bildschirmzeile 40 Zeichen Platz haben, werden auch 40 Pointeradressen gebraucht. Beim PAL-System erneuert der Video-Chip etwa 25mal pro Sekunde jede zweite Rasterlinie des 625zeiligen Fernsehschirms. Das amerikanische NTSC-System arbeitet mit 524 Zeilen, die pro Sekunde 30mal erneuert werden. Diese Abläufe verlangsamen die Arbeit des 6510 um 15 bis 20 Prozent.

Unterschiedlich aufgelöst

Die DMAs scheinen zunächst keinen Einfluß auf die internen Maschinenvorgänge zu haben, doch zeigt sich, daß das vorübergehende „Abschalten" des 6510 bei der Ein- und Ausgabe Probleme verursacht – beispielsweise beim Zugriff auf Cassettendateien. Dabei kann es sogar nötig werden, den Bildschirm während der E/A (mit POKE 53265,11) zu löschen und danach (mit POKE 53265,27) wieder anzuschalten. Ob der unmittelbare Speicherzugriff des VIC-II während der E/A Schwierigkeiten verursacht, hängt im wesentlichen vom externen Gerät und der Datenübertragungsmethode ab. Beim Diskettenzugriff braucht der Bildschirm nicht gelöscht zu werden.

Der Commodore 64 kann – umsichtig programmiert – zwei Grafikmodi mit hoher und niedriger Auflösung gleichzeitig auf dem Bildschirm darstellen. Damit lassen sich im oberen Teil beispielsweise Diagramme oder Grafik darstellen, während im unteren Teil Text erscheint oder Eingaben vorgenommen werden.

Unser Splitscreen-Programm richtet hinter dem ROM des BASIC-Interpreters einen Bildschirmspeicher für die hohe Auflösung ein und zeigt dessen Inhalt im oberen Bildschirmdrittel an. Der untere Teil des Schirms bleibt im normalen Textmodus.

In einem früheren Artikel wurde bereits gezeigt, wie die RAM-Vektoren des Commodore 64 eingesetzt werden. Über den Inhalt der Zwei-Byte-Pointer hatten wir den Ablauf von Betriebssystemroutinen auf unseren Programmcode geleitet. Mit dieser Technik lenken wir nun die IRQ-Routine um.

Es gibt mehrere Möglichkeiten, IRQs für den 6510 auszulösen. Im Normalfall setzt einer der CIA-Timer jede Sechzigstelsekunde ein Bit des Interrupt Flag Registers (IFR) bei Adresse 53273 ($D019). Damit wird eine IRQ-Bearbeitungsroutine ausgelöst, die unter anderem über die Matrix der Tastaturschalter abfragt, ob eine Taste gedrückt wurde. Bei bestimmten Ereignissen werden die IFR-Bits auch vom VIC-II Chip gesetzt – beispielsweise beim Zusammenstoß zweier Sprites oder wenn der Rasterzähler einen vorbestimmten Wert erreicht. Ein weiteres Register, das „Interrupt Enable Register" (IER) bei Adresse 53274 ($D01A), dient als „Schalter" für die IRQ-Leitung des 6510. Wenn im IER bestimmte Bits gesetzt werden, lösen die gesetzten Bits des IFR im 6510 ein Interrupt aus.

Hier der Aufbau von „Splitscreen":
1) Wenn am oberen Bildschirmrand ein Raster-Interrupt eintritt, wird der Bit-Map-Modus eingeschaltet und ein Raster-Interrupt auf die Bildschirmmitte gesetzt. Ein RTI gibt die Steuerung an das OS zurück.
2) Das Interrupt in der Bildschirmmitte schaltet wieder in die niedrige Auflösung, setzt den nächsten Taster-Interrupt auf den oberen Bildschirmrand und geht anschließend ins OS.

Bei diesem Aufbau wird der Teil des normalen Bildschirmspeichers, der dem oberen Bildschirmdrittel entspricht, für die Farbinformation des hohen Auflösungsmodus eingesetzt, während die restlichen zwei Drittel die normalen Zeichen enthalten.

Das Programm hat jedoch einen Haken. Raster-Interrupts treten alle Fünfzigstelsekunden ein und müssen sofort bearbeitet werden, da der Rasterstrahl sonst weiterwandert und falsche Daten anzeigt. Im Normalfall würde die IRQ-Bearbeitungsroutine jede Sechzigstelsekunde ausgelöst. Da die Routine jedoch recht lang ist, würde unser Programmkeil nur mit Verzögerung ausgeführt, wenn die Routine unmittelbar vor dem Raster-Interrupt aufgerufen wird. Diese Situation muß unbedingt vermieden werden, da die Grenzlinie zwischen den beiden Auflösungsarten sonst flackert.

Beschleunigtes BASIC

Wir lösen das Problem, indem wir die Ausführung der IRQ-Bearbeitungsroutine unmittelbar hinter den Programmkeil verlegen. Der Sechzigstelsekunden-Timer wird ausgeschaltet und so die normale IRQ-Auslösung verhindert. Sofort nach Ausführung unseres Programmkeils springen wir dann auf die IRQ-Bearbeitungsroutine. Dieser Vorgang sychronisiert die beiden Codes. Zwar wird nun die Tastatur seltener abgefragt, doch läuft dafür das BASIC auch merklich schneller.

Da der Bildschirmspeicher für hohe Auflösung nun hinter dem ROM des BASIC-Interpreters liegt, wird das PLOTten in BASIC komplizierter: Die Sprache kann den RAM-Bereich hinter dem BASIC-ROM nicht ansprechen und hat so keinen Zugang zu AND- und OR-Befehlen, die unter anderem für das Setzen der Pixel nötig sind. POKE umgeht jedoch das BASIC-ROM, spricht den dahinterliegenden RAM-Bereich unmittelbar an und macht so das PLOTten vom BASIC aus erst möglich. Bei reinen Maschinencodeprogrammen kann das BASIC-ROM völlig abgeschaltet werden.

Der erste Raster-Interrupt schaltet auf hohe Bildschirmauflösung.

Der zweite Raster-Interrupt veranlaßt das Zurückschalten in den Textmodus.

Viele Abenteuerspiele für den C 64 arbeiten mit Raster-Interrupts, um hochauflösende Bilder und Text gleichzeitig darzustellen. Der von Scott Adams entwickelte „Spiderman" zeigt im oberen Bildschirmdrittel die Spielszene, während der Rest der Anzeige die Beschreibung im normalen Textmodus enthält. Der Rasterlauf wurde so programmiert, daß er bei jedem Abtasten des Bildschirms zwei Interrupts auslöst: einen am oberen Bildschirmrand und den anderen etwa in Höhe des ersten Bildschirmdrittels. Bei jedem Raster-Interrupt wird eine Routine aufgerufen, die zwischen den beiden Darstellungsmodi umschaltet.

Bilderrahmen

In den nächsten beiden Artikeln sehen wir uns einige Fließkommaroutinen des Commodore 64 genauer an. Wir erstellen damit eine dreidimensionale Figur in hoher Auflösung.

Die Fließkommaroutinen des BASIC-Interpreters sind nicht sonderlich gut dokumentiert. So fehlt beispielsweise die Tabelle der Einsprungadressen, und einzelne Routinen lassen sich nur durch mühsames Experimentieren finden. Wir führen unsere Untersuchung daher anhand einer praktischen Aufgabe durch: Wir rotieren deshalb eine dreidimensionale Rahmenfigur auf dem hochauflösenden Bildschirm.

Die hier eingesetzten mathematischen Techniken lassen sich natürlich auch für andere Routinen einsetzen – beispielsweise bei der Matrixmultiplikation. Bedenken Sie aber, daß mathematische Berechnungen, die wegen der schnell aufeinanderfolgenden Bildschirmdarstellungen in Echtzeit programmiert werden, nicht unbedingt beste Programmiertechnik sind. Es ist eleganter, erst alle Bildschirmdaten zu berechnen und dann die Bewegungsfolge auszulösen.

BASIC-Variablen sind im Speicher über den BASIC-Programmen untergebracht. Sie befinden sich in einer Variablentabelle, auf deren Anfang die Speicherstellen 45 und 46 (dezimal) zeigen. Die Anfangsadresse der Variablentabelle wird so gefunden:

PEEK(45)+256*PEEK(46)

Unsere Tabelle zeigt verschiedene Variablenpointer mit ihrem normalen Inhalt.

Beachten Sie, daß dynamische Strings im Speicher von oben nach unten abgelegt werden, während Arrays über den anderen Variablen der Variablentabelle gespeichert sind. Taucht während der Programmausführung eine neue Variable auf, verschiebt das Betriebssystem den Arraybereich um die Anzahl Bytes nach oben, die die Variable benötigt.

Die Speicherung von Strings ist komplizierter. Ganzzahlenvariablen (außerhalb von Arrays) und Fließkommavariablen belegen je sieben Bytes, Stringvariablen jedoch bis zu 255 Bytes. Das Betriebssystem speichert daher nur die Stringlänge und einen Pointer auf die Anfangsadresse. Wenn in einem BASIC-Programm ein String definiert wird (etwa A$="ABCD"), dann zeigt der Pointer auf das erste Byte des Strings im BASIC-Programmbereich. Ein String dieser Art heißt „statisch". Wenn das Programm einen Stringwert ändert, wird der „dynamisch". Die Werte dynamischer Strings werden von der Speicherobergrenze an abwärts angelegt und die Pointer der Variablentabelle entsprechend verändert. Auf diese Weise hat jeder Eintrag die konstante Länge von sieben Bytes.

Pointer	Funktion	Normaler Inhalt
43/44	Anfang von BASIC	2049
45/46	Anfang der Variablen	je nach Programmlänge
47/48	Anfang der Arrays	je nach Variablenzahl
49/50	Ende der Arrays + 1	je nach Arrayzahl/-größe
51/52	Stringuntergrenze	je nach Stringzahl/-länge
55/56	Speicherobergrenze	40960

Die Adresse einer Variablen in der Variablentabelle läßt sich mit ein wenig Detektivarbeit leicht finden. Nach Aufruf der Variablen wird ihre Adresse in einen Pointer der Zero Page geladen (dezimal 71 und 72), der dort untersucht werden kann.

Das folgende Programm zeigt das Speicherformat der normalen Variablen. Die erste Routine holt einen String aus dem Speicher und sieht sich mit der gerade beschriebenen Technik den Inhalt der Adressen 71 und 72 an. Die Werte werden dann in zwei freien Bytes des Cassettenbuffers zur späteren Berechnung der Stringadresse gespeichert.

```
1000 REM**FIND A STRING IN MEMORY**
1010 X$="ABCDEF"
1020 REM**MAKE X$ CURRENT VARIABLE**
1030 X$=X$+""
1040 REM**SAVE POINTER TO VAR TABLE**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**ADDRESS IN VAR TABLE**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**LOOK AT ENTRY IN VAR TABLE**
1090 LS=PEEK(ADR):REM LENGTH OF STRING
1100 SA=PEEK(ADR+1)+256*PEEK(ADR+2)
1110 REM SA IS START ADDR OF STRING
1120 REM**NOW READ STRING**
1130 FORI=SATOSA+LS
1140 VAR$=VAR$+CHR$(PEEK(I))
1150 NEXT
1160 PRINTVAR$
```

Normalerweise benötigen Ganzzahlenvariablen (zum Beispiel X%) weniger Speicher und beschleunigen außerdem mathematische Abläufe. Beim C 64 ist das jedoch nicht der Fall. Wenn das Commodore-BASIC Ganzzahlberechnungen ausführt, wandelt es die Werte erst in das Fließkommaformat um und ruft dann die Fließkommaroutinen auf. Auf diese Weise belegen Ganzzahlvariablen, die eigentlich mit zwei Bytes auskommen, sieben Speicherbytes (falls sie nicht in Arrays abgelegt sind). Bei der Weiterverarbeitung werden die zusätzlichen Bytes jedoch ignoriert. So können Sie eine Ganzzahl im Speicher finden:

```
1000 REM**FIND AN INTEGER IN MEMORY**
1010 X%=3456
1020 REM**MAKE X% CURRENT VARIABLE**
1030 X%=X%
1040 REM**SAVE POINTER TO VAR TABLE**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**ADDRESS IN VAR TABLE**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**LOOK AT ENTRY IN VAR TABLE**
1090 LO=PEEK(ADR+1):HI=PEEK(ADR)
1100 REM**COMPUTE RESULT**
1110 SIGNBIT=(HIAND128)/128
1120 VAR=LO+256*(HIAND127)-32768*SIGNBIT
1130 PRINT VAR
```

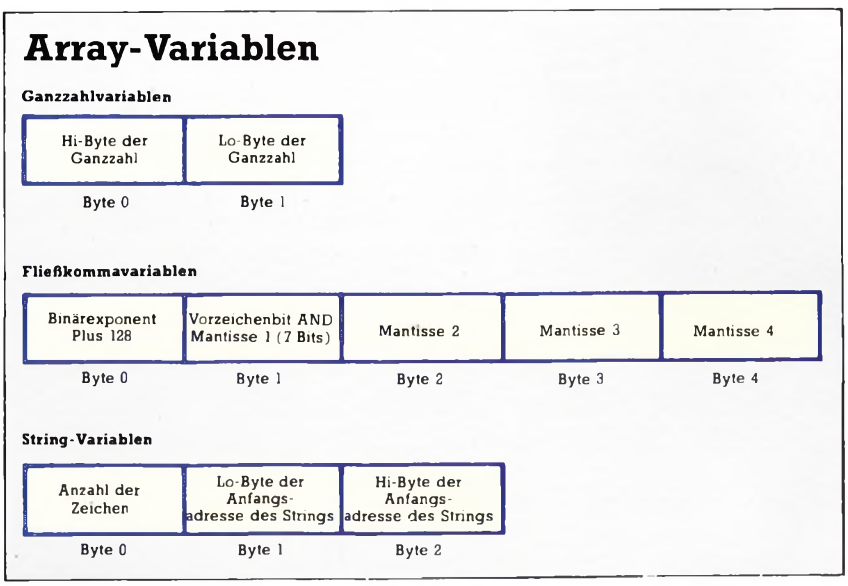
Mit dieser Technik können Sie natürlich auch Fließkommavariablen suchen. Es gibt jedoch eine weit praktischere Methode.

Das folgende Programm berechnet mit DEF FN die Adresse (in Speicherstelle 71 und 72) der aktuellen BASIC-Variablen. Da X eine Funktionsvariable ist, gibt die erzeugte Adresse die Speicherstelle des ersten Bytes von X in der BASIC-Variablentabelle wieder. Beim Aufruf von FN wird diese Adresse der Variablen ADD zugeordnet. Beachten Sie, daß die Übergabe von 0 (oder eines anderen Wertes) mit dem FN-Befehl den Wert von X in der Variablentabelle oder die von der Funktion berechnete Adresse nicht verändert.

Der Commodore 64 arbeitet mit drei Variablentypen, die mit verschiedenen Formaten im Speicher Platz finden. Ganzzahlvariablen werden in zwei Bytes als Zweierkomplement gespeichert. Fließkommazahlen verlangen fünf Bytes für Mantisse, Vorzeichenbit und Exponenten. Stringvariablen sind in einem anderen Bereich des Speichers untergebracht und belegen ein Byte pro Zeichen. Die Stringdaten befinden sich jedoch nicht in der Variablentabelle. Dort geben statt dessen drei Bytes die Zeichenzahl des Strings an, während eine 16-Bit-Adresse auf den Anfang des Strings im Speicher zeigt.

```
1000 REM**FIND A FPVAR IN MEMORY**
1010 DEF FNADR(X)=PEEK(71)+256*PEEK(72)
1020 ADD=FNADR(0):REM ALWAYS RETURNS ADDRESS OF X
1030 X=-3.14159
1040 REM**CONVERT FROM BASE 2 TO DECIMAL
1050 POWERTWO=2^(PEEK(ADD)-129)
1060 SIGN=(-1)^((PEEK(ADD+1)AND128)/128)
1070 REM*FRACTION PART IS 31 BITS WIDE*
1080 D1=PEEK(ADD+1)AND127:REM 7 BITS
1090 D2=PEEK(ADD+2):REM 8 BITS
1100 D3=PEEK(ADD+3):D4=PEEK(ADD+4)
1110 REM**GULP!**
1120 FRACT=2^(-7)*D1+2^(-15)*D2+2^(-23)*D3+2^
     (-31)*D4
1130 MANT=1+FRACT
1140 VAR=SIGN*POWERTWO*MANT
1150 PRINTVAR
```

Die Ausführung eines DIM-Befehls reserviert im Speicher Platz für ein Array. Der reservierte Speicher enthält einen Kopf und die Anzahl der für die Arrayelemente abgestellten Bytes. Das Format von Arrayelementen unterscheidet sich von allen anderen Variablentypen. Unser Diagramm zeigt die möglichen Variationen. Wenn Sie Arrayelemente vom Maschinencode aus ansprechen wollen, müssen Sie diese Unterschiede kennen.



Sehen wir uns nun die Fließkommaarithmetik genauer an. Bei der Ausführung von Fließkommaberechnungen speichert der BASIC-Interpreter seine Ergebnisse in zwei „Fließkommaakkumulatoren" – FAC und ARG –, die das gleiche Format haben wie die im Speicher abgelegten Variablen. FAC befindet sich bei den Adressen $61 bis $65 (dezimal 97 bis 101) und ARG bei $69 bis $6D (dezimal 105 bis 109). Aus Gründen der Einfachheit werden wir auch weiterhin mit den Interpreterroutinen arbeiten, die die Zahlen zwischen FAC und dem entsprechenden Speicher übertragen.

Hier sind die Routinen

Wir verwenden folgende Interpreterroutinen:

- MOVFM (CALL Adresse $BBA2):

MOVFM lädt den Inhalt von FAC mit der im Speicher abgelegten Fließkommavariablen. Symbolisch wird dieser Vorgang durch : ← M dargestellt. Für den Aufruf wird der Akkumulator mit dem Lo-Byte der Anfangsadresse der Speichervariablen geladen und das Y-Register mit dem Hi-Byte.

- MOVMF (CALL Adresse $BBD4):

Diese Routine legt den Inhalt von FAC in sieben Speicherbytes ab (M ← F). Für den Aufruf wird das X-Register mit dem Lo-Byte und Y mit dem Hi-Byte des Anfangsbytes der Variable im Speicher geladen.

- FMULT (CALL Adresse $BA28):

Diese Routine multipliziert den Inhalt von FAC mit einer zweiten Speichervariable und legt das Ergebnis in FAC ab. Die erste Variable wird mit MOVFM in FAC geladen. Wir zeigen auf die zweite Variable, indem wir vor Aufruf der Routine den Akkumulator mit dem Lo-Byte und Y mit dem Hi-Byte der Anfangsbytes laden. Falls nötig, können wir das Ergebnis mit MOVMF wieder speichern.

- FADD (CALL Adresse $B867):

Diese Additionsroutine führt FAC=MEM+FAC aus. Für den Aufruf wird der Akkumulator mit dem Lo-Byte von MEM und Y mit dem Hi-Byte von MEM geladen.

- FSUB (CALL Adresse $B850):

Diese Subtraktionsroutine führt FAC=MEM–FAC aus. Für den Aufruf wird der Akkumulator mit dem Lo-Byte von MEM und Y mit dem Hi-Byte von MEM geladen.

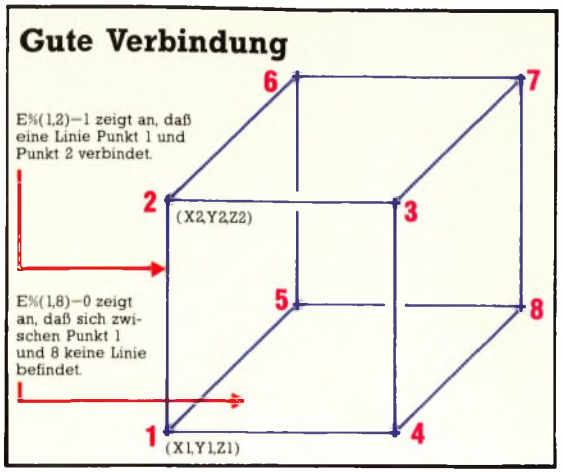
Diese Interpreterroutinen bauen wir in die erste Phase unseres Grafikprogramms ein.

Das Grafikprogramm arbeitet mit einer Umrißfigur, die durch eine Reihe von Punkten und eine Matrix von Eckverbindungen festgelegt wird. Die Punkte haben die Koordinaten X(I),Y(I),Z(I) – wobei I die Werte 1 bis NP (Anzahl der Punkte) durchlaufen kann. Die Matrix der Eckverbindungen ist E%(I,J) – wobei I und J die Zahlen 1 bis NP durchlaufen können. E%(I,J) wird als Eins definiert, wenn Punkt I mit

Punkt J verbunden ist. Anderenfalls ist diese Variable Null. Natürlich wird dabei der Speicher nicht optimal eingesetzt (wir arbeiten mit zwei Bytes, obwohl eines ausreichen würde). Unsere Methode hat jedoch den Vorteil, daß sich die zu verbindenden Punkte leichter angeben lassen. Außerdem kann der Wert von NP im praktischen Einsatz nie bemerkenswert hoch werden.

Um die Rotationsroutine im Maschinencode von BASIC aus ansprechen zu können, geben wir die Basisadressen der Arrays mit POKE in den Speicher. Wir beginnen mit Element 1 und müssen daher die Adresse von X(1),Y(1),Z(1) und E%(1,1) finden. Weiterhin müssen wir den Maschinencode mit mehreren Angaben versorgen: NP sowie COS und SIN des Rotationswinkels um die Achse Z.

Bei der Entwicklung eines komplexen Maschinencodeprogramms lohnt es sich, in kleinen Schritten vorzugehen, da sonst später viel Zeit für eine eventuelle Fehlersuche nötig ist. Wir haben das Projekt daher in drei Phasen geplant. Zunächst definieren wir die Algorithmen und schreiben ein Testprogramm in BASIC (BASIC-Rotationsprogramm). Es rotiert einen Würfel um die Z-Achse und bildet das Ergebnis auf den Ebenen X und Y ab.



Das Programm ROTSUB rotiert einen durchsichtigen Würfel in der hochauflösenden Grafik des Commodore 64. Die Linien der Figur werden mit vier Arrays definiert. Drei davon halten die X-, Y- und Z-Koordinaten jedes einzelnen Punktes. Das vierte Array – E%(,) – definiert, welche der Punkte durch Linien miteinander verbunden werden.

Unser Ziel ist es, dieses BASIC-Listing in Maschinencode zu übersetzen. Wir beschreiben zunächst jedoch eine Version der Subroutine bei Zeile 1800 im Maschinencode: das Programm I-ROTSUB. Der dritte Kasten enthält ein Programm, das I-ROTSUB testet.

In der nächsten Folge vervollständigen wir das Projekt über die Fließkommaroutinen und die dreidimensionale Figur in hoher Auflösung und drucken den restlichen Assemblercode und ein BASIC-Ladeprogramm ab.

Das Programm I-ROTSUB

Assemblieren und laden Sie das folgende Assemblerlisting und speichern Sie den Code unter I-ROT.HEX.

```
;++++++++++++++++++++++++++
;++     I-ROTSUB 64      ++
;++                      ++
;++++++++++++++++++++++++++
;++USES ARRAYS DEFINED   ++
;++FROM BASIC ADDRESSES  ++
;++MUST BE POKED FIRST   ++
;++++++++++++++++++++++++++
;
;PLTSUB = $C183
;XLO    = $C103
;XHI    = $C104
;YLO    = $C105
;
*       = $C544
;
;++++ ROTSUB  VARIABLES ++++
;
; VARIABLES CALLED FROM BASIC
XBASLO  *=*+1 ; POKE50500,X(0)LO
XBASHI  *=*+1 ; POKE50501,X(0)HI
YBASLO  *=*+1 ; POKE50502,Y(0)LO
YBASHI  *=*+1 ; POKE50503,Y(0)HI
NP      *=*+1 ; POKE50504,NP
; VARIABLES USED BY M/C
XILO    *=*+1
XIHI    *=*+1
YILO    *=*+1
YIHI    *=*+1
CSLO    *=*+1
CSHI    *=*+1
SNLO    *=*+1
SNHI    *=*+1
MEM1    *=*+5 ;FLOATING POINT VAR
MEM2    *=*+5 ;FLOATING POINT VAR
FAC     =$0061
ARG     =$0069
;
;INTERPRETER ARITHMETIC CALLS
;
FMULT   =$BA28 ; FAC=FAC*ARG
FADDT   =$B86A ; FAC=FAC+ARG
FSUB    =$B850 ; FAC=MEM-FAC
FADD    =$B867 ; FAC=FAC+MEM
MOVFM   =$BBA2 ; FAC=MEMORY
MOVMF   =$BBD4 ; MEMORY=FAC
;
;++++ SAVE REGISTERS++
;
        PHA
        TXA
        PHA
        TYA
        PHA
;
;++++ INITIALISE VARIABLES ++++
;
        LDA XBASLO
        STA XILO
        LDA XBASHI
        STA XIHI
        LDA YBASLO
        STA YILO
        LDA YBASHI
        STA YIHI
;
;++++PERFORM  MEM1=X(I)*CS-Y(I)*SN
;
START
        LDA XILO
        LDY XIHI
        JSR MOVFM   ; FAC = X(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT   ; FAC = X(I)*CS
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF   ; MEM1=X(I)*CS
        LDA YILO
        LDY YIHI
        JSR MOVFM   ; FAC = Y(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT   ; FAC = Y(I)*SN
        LDA #<MEM1
        LDY #>MEM1
        JSR FSUB    ; FAC = MEM1-FAC
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF   ; MEM1= FAC
;
;++++PERFORM MEM2=Y(I)*CN+X(I)*SN
;
        LDA YILO
        LDY YIHI
        JSR MOVFM   ; FAC = Y(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT   ; FAC = Y(I)*CS
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF   ; MEM2= Y(I)*CS
        LDA XILO
        LDY XIHI
        JSR MOVFM   ; FAC = X(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT   ; FAC = X(I)*SN
        LDA #<MEM2
        LDY #>MEM2
        JSR FADD    ; FAC = MEM2+FAC
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF   ; MEM2= FAC
;
;++++PERFORM X(I) =MEM1:Y(I)=MEM2
;
        LDA #<MEM1
        LDY #>MEM1
        JSR MOVFM   ; FAC =MEM1
        LDX XILO
        LDY XIHI
        JSR MOVMF   ; X(I)=FAC
        LDA #<MEM2
        LDY #>MEM2
        JSR MOVFM   ; FAC =MEM2
        LDX YILO
        LDY YIHI
        JSR MOVMF   ; Y(I)=FAC
;
;++++TEST END LOOP
;
        DEC NP
        BEQ EXIT
;
;++++INCREMENT ARRAY POINTERS
;
        LDA #$05
        CLC
        ADC XILO
        STA XILO
        BCC XNOHI
        INC XIHI
XNOHI
        LDA #$05
        CLC
        ADC YILO
        STA YILO
        BCC YNOHI
        INC YIHI
YNOHI
        JMP START
;
;++++ PULL REGISTERS OFF STACK ++++
;
EXIT
        PLA
        TAY
        PLA
        TAX
        PLA
;
        RTS
.END
```

Rotationsprogramm in BASIC

```
1000 REM** BASIC ROTATING CUBE**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 REM**DIMENSION ARRAYS**
1040 NP=8:REM NUMBER OF POINTS
1050 DIM X(NP),Y(NP),Z(NP)
1060 DIM ED(NP,NP):REM EDGE CONNECTIONS
1070 REM**INITIALISE ARRAYS**
1080 REM--CUBE COORDINATE DATA
1090 DATA  75, 75, 75:REM----------/1
1100 DATA -75, 75, 75:REM TOP FOUR /2
1110 DATA -75,-75, 75:REM POINTS   /3
1120 DATA  75,-75, 75:REM----------/4
1130 DATA  75, 75,-75:REM----------/5
1140 DATA -75, 75,-75:REM BOT FOUR /6
1150 DATA -75,-75,-75:REM POINTS   /7
1160 DATA  75,-75,-75:REM----------/8
1170 REM**ROTATE CUBE ABOUT X-AXIS π/4
1180 FORI=1TONP
1190 READX(I),Y(I),Z(I)
1200 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1210 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1220 NEXT
1230 REM**ROTATE CUBE ABOUT Z-AXIS π/4
1240 FORI=1TONP
1250 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1260 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1270 NEXT
1280 REM--EDGE CONNECTION DATA--
1290 E(1,2)=1:REM 1 CONNECTED TO 2
1300 E(2,3)=1:E(3,4)=1:E(4,1)=1
1310 E(5,6)=1:REM BOT SQUARE
1320 E(6,7)=1:E(7,8)=1:E(8,5)=5
1330 E(5,1)=1:REM TOP TO BOT EDGES
1340 E(6,2)=1:E(7,3)=1:E(8,4)=1
1350 REM**SYMMETRISE E(I,J)**
1360 FORI=1TONP:FORJ=1TONP
1370 IFE(I,J)=1THENE(J,I)=1
1380 NEXT:NEXT
1390 REM##########################
1400 REM##PLOT ROTATING CUBE##
1410 SA=2*π/45
1420 FOR A=π/4 TO π/4+2*π STEP SA
1430 GOSUB1800:REM ROTATE THRU SA
1440 GOSUB1590:REM INIT/CLEAR SCREEN
1450 REM--PLOT CUBE--
1460 FORI=1TONP
1470 FORJ=1TOI
1480 IF E(I,J)=0THEN1510:REM NOT JOINED
1490 GOSUB1630:REM COMPUTE PROJECTION
1500 GOSUB1670:REM JOIN POINTS
1510 NEXT:NEXT
1520 REM-------------
1530 NEXT A:REM NEXT ANGLE
1540 REM##########################
1550 REM**WAIT**
1560 GETA$:IFA$=""THEN1560
1570 GOSUB1760:REM RESET SCREEN
1580 END
1590 REM**SETUP HIRES**
1600 POKE49408,1:POKE49409,1
1610 POKE49410,1:SYS49422
1620 RETURN
1630 REM**COMPUTE PROJECTION ON HIRES**
1640 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1650 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1660 RETURN
1670 REM**LINESUB**
1680 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1690 MHI=INT(X1%/256):MLO=X1%-256*MHI
1700 NHI=INT(X2%/256):NLO=X2%-256*NHI
1710 POKE49920,MLO:POKE49921,MHI
1720 POKE49922,NLO:POKE49923,NHI
1730 POKE49924,Y1%:POKE49925,Y2%
1740 SYS49934:REM LINESUB
1750 RETURN
1760 REM**RESET SCREEN**
1770 POKE49408,0:SYS49422
1780 PRINTCHR$(147)
1790 RETURN
1800 REM**ROTATE CUBE ABOUT Z-AXIS/SA
1810 FORI=1TONP
1820 X(I)=X(I)*COS(SA)-Y(I)*SIN(SA)
1830 Y(I)=Y(I)*COS(SA)+X(I)*SIN(SA)
1840 NEXT
1850 RETURN
```

Testprogramm für I-ROTSUB

Geben Sie das folgende BASIC-Programm ein und speichern Sie es als TEST I-ROT. Achten Sie darauf, daß zwischen der Ausführung von Zeile 1880 bis 2000 und dem Aufruf von SYS50523 keine Variable definiert wird, da sonst die Gefahr des Programmabsturzes bestünde.

```
1000 REM** TEST I-ROT **
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"I-ROT.HEX",8,1
1040 REM**DIMENSION ARRAYS**
1050 NP=8:REM NUMBER OF POINTS
1060 DIM X(NP),Y(NP),Z(NP)
1070 DIM ED(NP,NP):REM EDGE CONNECTIONS
1080 REM**INITIALISE ARRAYS**
1090 REM--CUBE COORDINATE DATA
1100 DATA  75, 75, 75:REM----------/1
1110 DATA -75, 75, 75:REM TOP FOUR /2
1120 DATA -75,-75, 75:REM POINTS   /3
1130 DATA  75,-75, 75:REM----------/4
1140 DATA  75, 75,-75:REM----------/5
1150 DATA -75, 75,-75:REM BOT FOUR /6
1160 DATA -75,-75,-75:REM POINTS   /7
1170 DATA  75,-75,-75:REM----------/8
1180 REM**ROTATE SPACE ABOUT X-AXIS π/4
1190 FORI=1TONP
1200 READX(I),Y(I),Z(I)
1210 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1220 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1230 NEXT
1240 REM**ROTATE SPACE ABOUT Z-AXIS π/4
1250 FORI=1TONP
1260 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1270 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1280 NEXT
1290 REM--EDGE CONNECTION DATA--
1300 E(1,2)=1:REM 1 CONNECTED TO 2
1310 E(2,3)=1:E(3,4)=1:E(4,1)=1
1320 E(5,6)=1:REM BOT SQUARE
1330 E(6,7)=1:E(7,8)=1:E(8,5)=5
1340 E(5,1)=1:REM TOP TO BOT EDGES
1350 E(6,2)=1:E(7,3)=1:E(8,4)=1
1360 REM**SYMMETRISE E(I,J)**
1370 FORI=1TONP:FORJ=1TONP
1380 IFE(I,J)=1THENE(J,I)=1
1390 NEXT:NEXT
1400 REM##########################
1410 REM##PLOT ROTATING CUBE##
1420 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1430 FOR A=0 TO 2*π STEP SA
1440 GOSUB1870:REM ROTATE THRU SA
1450 GOSUB1600:REM INIT/CLEAR SCREEN
1460 REM--PLOT CUBE--
1470 FORI=1TONP
1480 FORJ=1TOI
1490 IF E(I,J)=0THEN1520:REM NOT JOINED
1500 GOSUB1640:REM COMPUTE PROJECTION
1510 GOSUB1680:REM JOIN POINTS
1520 NEXT:NEXT
1530 REM-------------
1540 NEXT A:REM NEXT ANGLE
1550 REM##########################
1560 REM**WAIT**
1570 GETA$:IFA$=""THEN1570
1580 GOSUB1770:REM RESET SCREEN
1590 END
1600 REM**SETUP HIRES**
1610 POKE49408,1:POKE49409,1
1620 POKE49410,1:SYS49422
1630 RETURN
1640 REM**COMPUTE PROJECTION ON HIRES**
1650 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1660 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1670 RETURN
1680 REM**LINESUB**
1690 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1700 MHI=INT(X1%/256):MLO=X1%-256*MHI
1710 NHI=INT(X2%/256):NLO=X2%-256*NHI
1720 POKE49920,MLO:POKE49921,MHI
1730 POKE49922,NLO:POKE49923,NHI
1740 POKE49924,Y1%:POKE49925,Y2%
1750 SYS49934:REM LINESUB
1760 RETURN
1770 REM**RESET SCREEN**
1780 POKE49408,0:SYS49422
1790 PRINTCHR$(147)
1800 RETURN
1810 REM**ROTATE CUBE ABOUT Z-AXIS/SA
1820 FORI=1TONP
1830 X(I)=X(I)*CS-Y(I)*SN
1840 Y(I)=Y(I)*CS+X(I)*SN
1850 NEXT
1860 RETURN
1870 REM**ROTATE ABOUT Z-AXIS/SA
1880 X(1)=X(1):REM X(1) CURRENT VAR
1890 POKE50500,PEEK(71):REM X(1)LO
1900 POKE50501,PEEK(72):REM X(1)HI
1910 Y(1)=Y(1):REM Y(1) CURRENT VAR
1920 POKE50502,PEEK(71):REM Y(1)LO
1930 POKE50503,PEEK(72):REM Y(1)HI
1940 POKE50504,NP:REM NUMBER OF POINTS
1950 CS=CS:REM MAKE CS CURRENT VAR
1960 POKE50509,PEEK(71)
1970 POKE50510,PEEK(72)
1980 SN=SN:REM MAKE SN CURRENT VAR
1990 POKE50511,PEEK(71)
2000 POKE50512,PEEK(72)
2010 SYS50523
2020 RETURN
```

Grafik mit Dreh

Im ersten Artikel über die dreidimensionale Grafik des Commodore 64 hatten wir einen Teil des Grafikprogramms in Maschinensprache übertragen. Wir übersetzen nun den Rest des BASIC-Listings und stellen weitere Fließkommaroutinen vor.

Die Programmkombination Test I-Rot und I-Rot-Hex läuft schon recht schnell, wird aber durch das Abfragen der Matrix E%(I,J) stark verlangsamt. Die Matrix legt fest, welche Punkte der Figur miteinander verbunden sind und gibt uns damit die Information, welche Linien gezeichnet werden sollen. Um höhere Ausführungsgeschwindigkeiten zu erreichen, müssen wir daher auch den Rest des BASIC-Programms in Maschinencode übersetzen.

Für die mathematischen Berechnungen in den Zeilen 1640 und 1650

```
X1%=X(I)+159:Y1%=199-(Z(I)+100)
X2%=X(J)+159:Y2%=199-(Z(J)+100)
```

setzen wir weitere Interpreterroutinen ein. Beide BASIC-Zeilen nehmen eine Fließkommavariable, beispielsweise X(I), addieren (im Fließkommaformat) 159, interpretieren das Ergebnis als Ganzzahl und speichern es im Zwei-Byte-Format als X1%. Wir ersetzen sie mit

- FLPINT (CALL Adresse $B1AA):

FLPINT nimmt den Ganzzahlenwert der in FAC gespeicherten Zahl und gibt das Ergebnis (falls es zwischen −32767 und 32767 liegt) an die Register Y und A im Lo-Hi-Format zurück. Beachten Sie, daß im Gegensatz zu den meisten anderen Interpreterroutinen hier das Format Lo-Hi eingesetzt ist.

- SNGFT (CALL Adresse $B3A2):

SNGFT nimmt eine Ganzzahl im Ein-Byte-Format aus dem Y-Register und legt sie im Fließkommaformat in FAC ab.

SNGFT wird in der Subroutine SETUP des Assemblerlistings (Zeile 5150) eingesetzt. Wenn beispielsweise der Dezimalwert 159 in das Indexregister Y gesetzt wird, wandelt ein Aufruf von SNGFT diesen Wert um und legt das Ergebnis in FAC ab. MOVFM überträgt den Inhalt von FAC dann in MEM1.

Für die Umwandlung des BASIC-Programms in Maschinencode müssen wir nun festlegen, welche Arrayelemente den rotierenden Umriß definieren sollen. Dabei kann die Berechnung der Arraypointer recht schwierig werden. Die Koordinatenarrays X(I), Y(I) und Z(I) machen keine Probleme, da wir in jedem Array nur jeweils fünf Bytes auf den Pointer addieren müssen, um die Adresse des nächsten Elements zu erhalten.

Das zweidimensionale Array E%(I,J) ist jedoch völlig anders aufgebaut. Es hat folgendes Format:

```
E%(0,0)E%(1,0)E%(2,0) . . .E%(NP,0)
E%(0,1)E%(1,1)E%(2,1) . . .E%(NP,1)
etc. . . .
```

Das Array besteht aus Speicherblöcken, die je 2*(NP−1) lang sind, wobei jeder Block den Werten des zweiten definierten Elements entspricht und jedes Element (des Ganzzahlarrays) genau zwei Bytes lang ist.

Da wir den BASIC-Code genau in den Maschinencode übertragen wollen, lauten die I,J Schleifen zum Durchsuchen von E%(I,J):

```
FOR I=1 TO NP
FOR J = 1 TO I
```

Nun geht's rund

Eine Pointerveränderung, die im Maschinencode dem BASIC-Befehl NEXT I entspricht, verkompliziert sich jedoch, wenn es möglich sein soll, alle Elemente mit der Definition Null zu überspringen. Für die Rotation der Figur werden die Elemente von E%(I,J) daher in folgender Reihenfolge angesprochen:

```
E%(1,1)
E%(1,2)E%(2,2)
E%(1,3)E%(2,3)E%(3,3)
E%(1,4))E%(2,4)E%(3,4)E%(4,4) etc.
```

Ein schneller Überschlag zeigt, daß 2*(NP+1) bei jeder Inkrementierung von I auf den Pointer addiert werden muß. Im Maschinencode des 6502 wird dafür am besten die indirekte Adressierung eingesetzt. Hier der Code:

```
LDY JINDEX
LDA (ZPTEMP),Y
```

ZPTEMP ist ein Zwei-Byte Pointer auf die Zero Page. JINDEX steuert J. Bei jeder Inkrementierung von J muß nun auch ZPTEMP erhöht werden. Durch die Inkrementierung von ZPTEMP und Y erhöht sich das „Netto"-Offset bei jeder Inkrementierung von J um zwei Bytes. ZPTEMP wird daher bei jeder Wiederholung der I-Schleife um (2*NP+1)−(I−1) inkrementiert. (I-1) muß von der Blocklänge abgezogen werden, da ZPTEMP bei der (gerade beendeten) J-Schleife bereits (I−1)mal inkrementiert wurde. Wenn das Offset über die Formel errechnet wird, zeigt ZPTEMP nach jeder Erhöhung von I auf das richtige Byte.

Es wäre praktisch, wenn eine Interpreterroutine die Variable E%(I,J) für uns finden könnte. Tatsächlich gibt es eine derartige Routine. Sie ist jedoch extrem kompliziert und langsam.

Routinen zum Rotieren

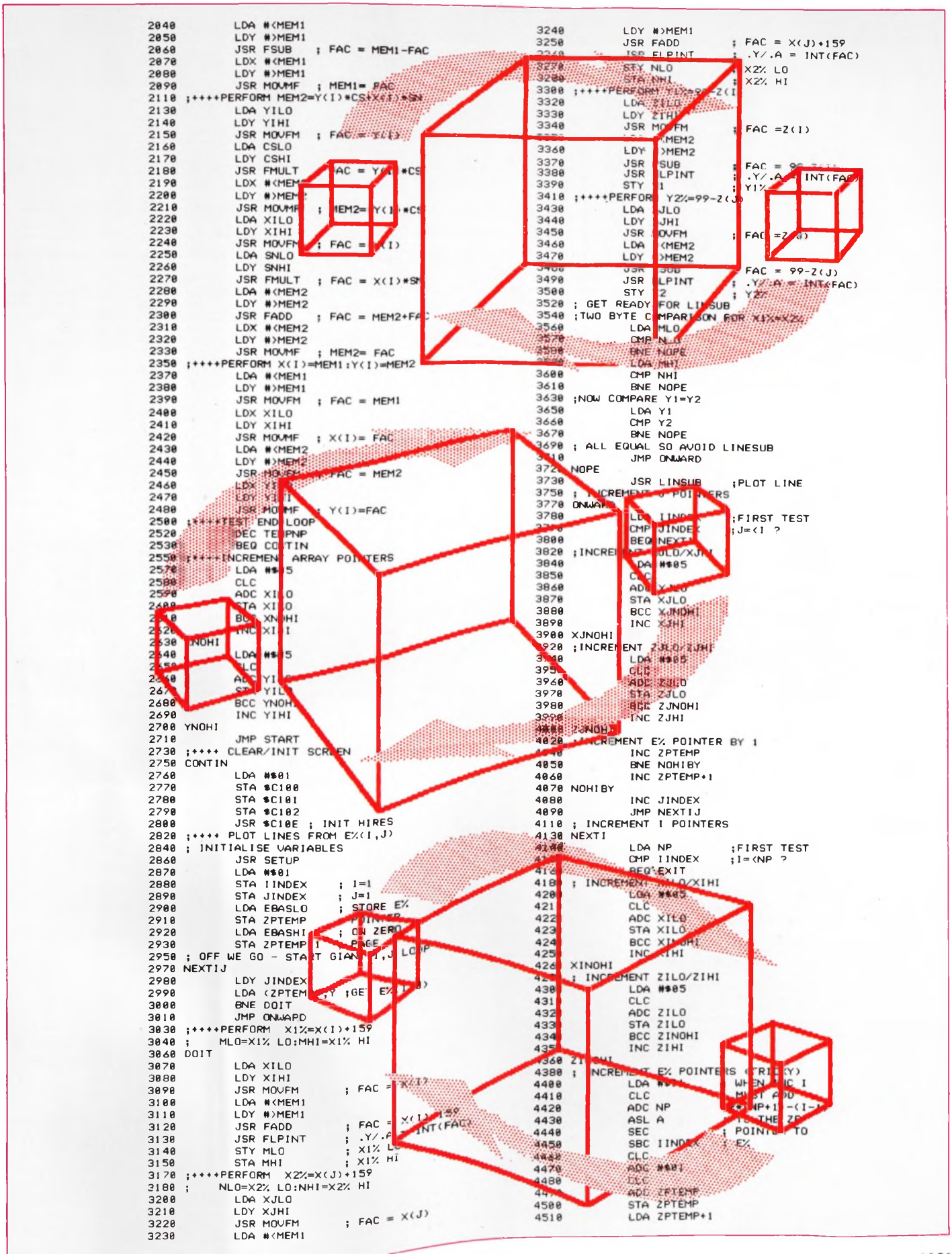
Das folgende Maschinencodeprogramm rotiert auf dem hochauflösenden Bildschirm die im Array E%(I,J) definierte Umrißfigur. Wir haben damit eine Maschinencodeversion der BASIC-Schleife, die die Arrays durcheucht und so die rotierende Figur erzeugt. Das Ladeprogramm enthält den Quelltext in BASIC DATA-Befehlen zusammen mit dem BASIC-Testprogramm legt es die Maschinencodevariablen an und ruft die Routine auf. Das Testprogramm arbeitet entweder mit der assemblierten Version des Quellencode – II-ROT.HEX oder mit dem vom Ladeprogramm erzeugten Code. Wenn Sie mit dem Ladeprogramm arbeiten, sollten Sie allerdings die Zeile 1030 des Testprogramms löschen.

BASIC-Ladeprogramm

```
1000 REM**  INSERT II/C II-ROT.HEX  **
1010 REM*****************************
1020 DATA72,138,72,152,72,32,101,159
1030 DATA173,83,197,172,84,197,32,162
1040 DATA187,173,75,197,172,76,197,32
1050 DATA40,186,162,94,160,197,32,212
1060 DATA187,173,87,197,172,88,197,32
1070 DATA162,187,173,77,197,172,78,197
1080 DATA32,40,186,169,94,160,197,32,8
1090 DATA184,162,94,160,197,32,212,187
1100 DATA173,87,197,172,88,197,32,162
1110 DATA187,173,75,197,172,76,197,32
1120 DATA40,186,162,99,160,197,32,212
1130 DATA187,173,83,197,172,84,197,32
1140 DATA162,187,173,77,197,172,78,197
1150 DATA32,40,186,169,99,160,197,32
1160 DATA183,184,162,99,160,197,32,212
1170 DATA187,169,94,160,187,32,212,187
1180 DATA174,83,197,172,84,197,32,212
1190 DATA187,169,99,160,197,32,162,187
1200 DATA174,87,197,172,88,197,32,212
1210 DATA187,201,99,197,240,31,169,5,24
1220 DATA109,87,197,141,83,197,144,3
1230 DATA238,84,197,169,5,24,109,87,197
1240 DATA141,87,197,144,3,238,88,197,76
1250 DATA112,197,169,1,141,0,193,141,1
1260 DATA193,169,2,193,32,14,193,32,161
1270 DATA199,169,1,141,81,197,141,82
1280 DATA197,173,79,197,133,253,173,80
1290 DATA197,133,254,172,82,197,177,253
1300 DATA208,3,76,207,198,173,83,197
1310 DATA174,84,197,32,162,187,169,94
1320 DATA160,197,32,103,184,32,170,177
1330 DATA140,0,195,141,1,195,173,85,197
1340 DATA172,86,197,32,162,187,169,94
1350 DATA160,197,32,103,184,32,170,177
1360 DATA140,2,195,141,3,195,173,89,197
1370 DATA172,90,197,32,162,187,169,99
1380 DATA160,197,32,80,184,32,170,177
1390 DATA140,4,195,173,91,197,172,92
1400 DATA197,32,162,187,169,99,160,197
1410 DATA32,80,184,32,170,177,140,5,195
1420 DATA173,0,195,205,2,195,208,13,173
1430 DATA1,195,205,3,195,208,1,173,4
1440 DATA195,205,5,195,208,3,76,207,198
1450 DATA32,14,195,173,81,197,205,82
1460 DATA197,240,40,169,5,24,109,85,197
1470 DATA141,85,197,144,3,238,86,197
1480 DATA169,5,24,109,91,197,141,91,197
1490 DATA144,3,238,92,197,230,253,208,2
1500 DATA230,254,238,82,197,76,73,198
1510 DATA173,74,197,205,81,197,240,88
1520 DATA169,5,24,109,83,197,141,89,197
1530 DATA144,3,238,84,197,169,5,24,109
1540 DATA89,197,141,89,197,144,3,238,90
1550 DATA197,169,1,24,109,74,197,10,56
1560 DATA237,81,197,24,105,1,201,101,253
1570 DATA133,253,165,254,105,0,133,254
1580 DATA173,68,197,141,85,197,173,69
1590 DATA197,141,86,197,173,72,197,141
1600 DATA91,197,173,73,197,141,92,197
1610 DATA169,1,141,82,197,141,81,197,76
1620 DATA73,198,104,168,104,170,104,96
1630 DATA173,68,197,141,83,197,141,85
1640 DATA197,173,69,197,141,84,197,141
1650 DATA86,197,173,70,197,141,87,197
1660 DATA173,71,197,141,88,197,173,74
1670 DATA197,141,93,197,173,72,197,141
1680 DATA89,197,141,91,197,173,73,197
1690 DATA141,90,197,141,92,197,160,159
1700 DATA32,162,179,162,94,160,197,32
1710 DATA212,187,160,99,32,162,179,162
1720 DATA99,160,197,32,212,187,96
1730 DATA79160:REM*CHECKSUM*
1740 FORI=50536TO51123
1750 READX:POKEI,X:CC=CC+X
1760 NEXT
1770 READX:IFX<>CCTHENPRINT"CHECKSUM ERROR":STOP
1780 PRINT"II-ROT.HEX INSTALLED OK"
```

Assembler-Listing

```
1010 ;++++++++++++++++++++++++++++
1020 ;++        ROTATE 3D        ++
1030 ;++                         ++
1040 ;++++++++++++++++++++++++++++
1140 ;
1150 *       = $C544
1190 ; VARIABLES CALLED FROM BASIC
1200 ;
1210 XBASLO  *=*+1 ; POKE50500,X(1)LO
1220 XBASHI  *=*+1 ; POKE50501,X(1)HI
1230 YBASLO  *=*+1 ; POKE50502,Y(1)LO
1240 YBASHI  *=*+1 ; POKE50503,Y(1)HI
1250 ZBASLO  *=*+1 ; POKE50504,Z(1)LO
1260 ZBASHI  *=*+1 ; POKE50505,Z(1)HI
1270 NP      *=*+1 ; POKE50506,NP
1280 CSLO    *=*+1 ; POKE50507,CSLO
1290 CSHI    *=*+1 ; POKE50508,CSHI
1300 SNLO    *=*+1 ; POKE50509,SNLO
1310 SNHI    *=*+1 ; POKE50510,SNHI
1320 EBASLO  *=*+1 ; POKE50511,E%(1,1)LO
1330 EBASHI  *=*+1 ; POKE50512,E%(1,1)HI
1340 ;
1350 ; VARIABLES USED BY II/C
1370 INDEX   *=*+1
1380 JINDEX  *=*+1
1390 XILO    *=*+1
1400 XIHI    *=*+1
1410 XJLO    *=*+1
1420 XJHI    *=*+1
1430 YILO    *=*+1
1440 YIHI    *=*+1
1450 ZILO    *=*+1
1460 ZIHI    *=*+1
1470 ZJLO    *=*+1
1480 ZJHI    *=*+1
1490 TEMPNP  *=*+1
1500 MEM1    *=*+5 ;FLOATING POINT VAR
1510 MEM2    *=*+5 ;FLOATING POINT VAR
1520 ZPTEMP  =$FD  ;FREE ZERO PAGE LOC
1530 ;
1540 ;INTERPRETER ARITHMETIC CALLS
1560 FMULT   =$BA28 ; FAC=FAC*MEM
1570 FSUB    =$B850 ; FAC=MEM-FAC
1580 FADD    =$B867 ; FAC=FAC+MEM
1590 MOVFM   =$BBA2 ; FAC=MEM
1600 MOVMF   =$BBD4 ; MEM=FAC
1610 FLPINT  =$B1AA ; .Y/.A=INT(FAC) N.B. HI/LO ORDER !!
1620 SNGFT   =$B3A2 ; FAC= .Y
1630 ;
1640 ;OTHER MACHINE CODE ROUTINE/VARS
1660 LINSUB  =$C30E
1670 M%0     =$C300
1680 M%1     =$C301
1690 NLO     =$C302
1700 NHI     =$C303
1710 Y1      =$C304
1720 Y2      =$C305
1730 ;++++ SAVE REGISTERS++
1740         PHA
1750         TXA
1760         PHA
1770         TYA
1780         PHA
1830 ;++++ INITIALISE VARIABLES ++++
1840         JSR SETUP
1850 ;++++PERFORM  MEM1=X(I)*CS-Y(I)*SN
1860 START
1870         LDA XILO
1900         LDY XIHI
1910         JSR MOVFM  ; FAC = X(I)
1920         LDA CSLO
1930         LDY CSHI
1940         JSR FMULT  ; FAC = X(I)*CS
1950         LDX #<MEM1
1960         LDY #>MEM1
1970         JSR MOVMF  ; MEM1=X(I)*CS
1980         LDA YILO
1990         LDY YIHI
2000         JSR MOVFM  ; FAC = Y(I)
2010         LDA SNLO
2020         LDY SNHI
2030         JSR FMULT  ; FAC = Y(I)*SN
```



```
2040          LDA #<MEM1
2050          LDY #>MEM1
2060          JSR FSUB    ; FAC = MEM1-FAC
2070          LDX #<MEM1
2080          LDY #>MEM1
2090          JSR MOVMF   ; MEM1= FAC
2110 ;++++PERFORM MEM2=Y(I)*CS+X(I)*SN
2130          LDA YILO
2140          LDY YIHI
2150          JSR MOVFM   ; FAC = Y(I)
2160          LDA CSLO
2170          LDY CSHI
2180          JSR FMULT   ; FAC = Y(I)*CS
2190          LDX #<MEM
2200          LDY #>MEM
2210          JSR MOVMF   ; MEM2= Y(I)*CS
2220          LDA XILO
2230          LDY XIHI
2240          JSR MOVFM   ; FAC = X(I)
2250          LDA SNLO
2260          LDY SNHI
2270          JSR FMULT   ; FAC = X(I)*SN
2280          LDA #<MEM2
2290          LDY #>MEM2
2300          JSR FADD    ; FAC = MEM2+FAC
2310          LDX #<MEM2
2320          LDY #>MEM2
2330          JSR MOVMF   ; MEM2= FAC
2350 ;++++PERFORM X(I)=MEM1:Y(I)=MEM2
2370          LDA #<MEM1
2380          LDY #>MEM1
2390          JSR MOVFM   ; FAC = MEM1
2400          LDX XILO
2410          LDY XIHI
2420          JSR MOVMF   ; X(I)= FAC
2430          LDA #<MEM2
2440          LDY #>MEM2
2450          JSR MOVFM   ; FAC = MEM2
2460          LDX YI
2470          LDY YI
2480          JSR MOVMF   ; Y(I)=FAC
2500 ;++++TEST END LOOP
2520          DEC TEMPNP
2530          BEQ CONTIN
2550 ;++++INCREMENT ARRAY POINTERS
2570          LDA #$05
2580          CLC
2590          ADC XILO
2600          STA XILO
2610          BCC XNOHI
2620          INC XIHI
2630 XNOHI
2640          LDA #$05
2650          CLC
2660          ADC YILO
2670          STA YILO
2680          BCC YNOHI
2690          INC YIHI
2700 YNOHI
2710          JMP START
2730 ;++++ CLEAR/INIT SCREEN
2750 CONTIN
2760          LDA #$01
2770          STA $C100
2780          STA $C101
2790          STA $C102
2800          JSR $C10E ; INIT HIRES
2820 ;++++ PLOT LINES FROM E%(I,J)
2840 ; INITIALISE VARIABLES
2860          JSR SETUP
2870          LDA #$01
2880          STA IINDEX     ; I=1
2890          STA JINDEX     ; J=1
2900          LDA EBASLO     ; STORE E%
2910          STA ZPTEMP     POINTER
2920          LDA EBASHI     ; ON ZERO
2930          STA ZPTEMP+1   ; PAGE
2950 ; OFF WE GO - START GIANT I,J LOOP
2970 NEXTIJ
2980          LDY JINDEX
2990          LDA (ZPTEMP),Y ;GET E%(I,J)
3000          BNE DOIT
3010          JMP ONWARD
3030 ;++++PERFORM  X1%=X(I)+159
3040 ;    MLO=X1% LO:MHI=X1% HI
3060 DOIT
3070          LDA XILO
3080          LDY XIHI
3090          JSR MOVFM       ; FAC = X(I)
3100          LDA #<MEM1
3110          LDY #>MEM1
3120          JSR FADD        ; FAC = X(I)+159
3130          JSR FLPINT      ; .Y/.A = INT(FAC)
3140          STY MLO         ; X1% LO
3150          STA MHI         ; X1% HI
3170 ;++++PERFORM  X2%=X(J)+159
3180 ;    NLO=X2% LO:NHI=X2% HI
3200          LDA XJLO
3210          LDY XJHI
3220          JSR MOVFM       ; FAC = X(J)
3230          LDA #<MEM1
3240          LDY #>MEM1
3250          JSR FADD        ; FAC = X(J)+159
3260          JSR FLPINT      ; .Y/.A = INT(FAC)
3270          STY NLO         ; X2% LO
3280          STA NHI         ; X2% HI
3300 ;++++PERFORM Y1%=99-Z(I)
3320          LDA ZILO
3330          LDY ZIHI
3340          JSR MOVFM       ; FAC =Z(I)
3350          LDA #<MEM2
3360          LDY #>MEM2
3370          JSR FSUB        ; FAC = 99-Z(I)
3380          JSR FLPINT      ; .Y/.A = INT(FAC)
3390          STY Y1          ; Y1%
3410 ;++++PERFORM Y2%=99-Z(J)
3430          LDA ZJLO
3440          LDY ZJHI
3450          JSR MOVFM       ; FAC =Z(J)
3460          LDA #<MEM2
3470          LDY #>MEM2
3480          JSR FSUB        ; FAC = 99-Z(J)
3490          JSR FLPINT      ; .Y/.A = INT(FAC)
3500          STY Y2          ; Y2%
3520 ; GET READY FOR LINSUB
3540 ;TWO BYTE COMPARISON FOR X1%=X2%
3560          LDA MLO
3570          CMP NLO
3580          BNE NOPE
3590          LDA MHI
3600          CMP NHI
3610          BNE NOPE
3630 ;NOW COMPARE Y1=Y2
3650          LDA Y1
3660          CMP Y2
3670          BNE NOPE
3690 ; ALL EQUAL SO AVOID LINESUB
3710          JMP ONWARD
3720 NOPE
3730          JSR LINSUB      ;PLOT LINE
3750 ; INCREMENT J POINTERS
3770 ONWARD
3780          LDA JINDEX      ;FIRST TEST
3790          CMP JINDEX      ;J=<I ?
3800          BEQ NEXTI
3820 ;INCREMENT XJLO/XJHI
3840          LDA #$05
3850          CLC
3860          ADC XJLO
3870          STA XJLO
3880          BCC XJNOHI
3890          INC XJHI
3900 XJNOHI
3920 ;INCREMENT ZJLO/ZJHI
3940          LDA #$05
3950          CLC
3960          ADC ZJLO
3970          STA ZJLO
3980          BCC ZJNOHI
3990          INC ZJHI
4000 ZJNOHI
4020 ;INCREMENT E% POINTER BY 1
4040          INC ZPTEMP
4050          BNE NOHIBY
4060          INC ZPTEMP+1
4070 NOHIBY
4080          INC JINDEX
4090          JMP NEXTIJ
4110 ; INCREMENT I POINTERS
4130 NEXTI
4140          LDA NP          ;FIRST TEST
4150          CMP IINDEX      ;I=<NP ?
4160          BEQ EXIT
4180 ; INCREMENT XILO/XIHI
4200          LDA #$05
4210          CLC
4220          ADC XILO
4230          STA XILO
4240          BCC XINOHI
4250          INC XIHI
4260 XINOHI
4270 ; INCREMENT ZILO/ZIHI
4300          LDA #$05
4310          CLC
4320          ADC ZILO
4330          STA ZILO
4340          BCC ZINOHI
4350          INC ZIHI
4360 ZINOHI
4380 ; INCREMENT E% POINTERS (TRICKY)
4400          LDA #$01        ; WHEN INC I
4410          CLC             ; MUST ADD
4420          ADC NP          ; (NP+1)-(I+1)
4430          ASL A           ; TO THE ZP
4440          SEC             ; POINTER TO
4450          SBC IINDEX      ; E%
4460          CLC
4470          ADC #$01
4480          CLC
4490          ADC ZPTEMP
4500          STA ZPTEMP
4510          LDA ZPTEMP+1
```

```
4520          ADC #$00
4530          STA ZPTEMP+1
4550 ; REINITIALISE XJLO/XJHI
4570          LDA XBASLO
4580          STA XJLO
4590          LDA XBASHI
4600          STA XJHI
4620 ; REINITIALISE ZJLO/ZJHI
4640          LDA ZBASLO
4650          STA ZJLO
4660          LDA ZBASHI
4670          STA ZJHI
4690 ; REINITIALISE JINDEX
4710          LDA #$01
4720          STA JINDEX
4740 ; INCREMENT IINDEX
4760          INC IINDEX
4770          JMP NEXT J
4790 ; GIANT LOOP ENDS HERE
4800 ;
4810 ;++++ PULL REGISTERS ++++
4820 ;
4830 EXIT
4840          PLA
4850          TAY
4860          PLA
4870          TAX
4880          PLA
4890          RTS
4930 ;++++ SUBROUTINE
4940 ;
4950 SETUP
4960          LDA XBASLO
4970          STA XILO
4980          STA XJLO
4990          LDA XBASHI
5000          STA XIHI
5010          STA XJHI
5020          LDA YBASLO
5030          STA YILO
5040          LDA YBASHI
5050          STA YIHI
5060          LDA NP
5070          STA TEMPNP
5080          LDA ZBASLO
5090          STA ZILO
5100          STA ZJLO
5110          LDA ZBASHI
5120          STA ZIHI
5130          STA ZJHI
5140          LDY #159
5150          JSR SNGFT     ; FAC = 159
5160          LDX #<MEM1
5170          LDY #>MEM1
5180          JSR MOVMF     ; MEM1=159 IN FPT
5190          LDY #99
5200          JSR SNGFT     ; FAC = 99
5210          LDX #<MEM2
5220          LDY #>MEM2
5230          JSR MOVMF     ; MEM2=99 IN FPT
5240          RTS
```

Testprogramm in BASIC

```
1000 REM**PLOT ROTATING CUBES**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"11-ROT.HEX",8,1
1040 PRINTCHR$(147)"WAIT 17 SECS"
1050 REM**DIMENSION ARRAYS**
1060 NP=24:REM NUMBER OF POINTS
1070 DIM X(NP),Y(NP),Z(NP)
1080 DIM E%(NP,NP):REM EDGE CONNECTIONS
1090 REM**INITIALISE ARRAYS**
1100 REM--INITIAL CUBE COORDINATE DATA
1110 DATA  50, 50, 50:REM----------1
1120 DATA -50, 50, 50:REM TOP FOUR 2
1130 DATA -50,-50, 50:REM POINTS   3
1140 DATA  50,-50, 50:REM----------4
1150 DATA  50, 50,-50:REM----------5
1160 DATA -50, 50,-50:REM BOT FOUR 6
1170 DATA -50,-50,-50:REM POINTS   7
1180 DATA  50,-50,-50:REM----------8
```

Der folgende Programmteil liest die zuvor definierten Daten in die Arrays X(), Y() und Z(). Beachten Sie, daß die Anfangswerte der kleinen Würfel um ein Drittel verringert wurden.

```
1190 REM**READ IN DATA FOR 3-CUBES**
1200 FORI=1TO8:REM CENTRE BIG CUBE
1210 READX(I),Y(I),Z(I)
1220 NEXT
1230 RESTORE
1240 FORI=9TO16:REM RIGHT SMALL CUBE
1250 READX,Y,Z
1260 X(I)=.3*X+120:Y(I)=.3*Y:Z(I)=.3*Z
1270 NEXT
1280 RESTORE
1290 FORI=17TO24:REM LEFT SMALL CUBE
1300 READX,Y,Z
1310 X(I)=.3*X-120:Y(I)=.3*Y:Z(I)=.3*Z
1320 NEXT
```

Der nächste Abschnitt rotiert die Koordinaten der Punkte, so daß die drei Würfel unter einem Winkel von 45 Grad erscheinen.

```
1330 REM**ROTATE SPACE ABOUT X-AXIS 45
1340 FORI=1TONP
1350 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1360 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1370 NEXT
1380 REM**ROTATE SPACE ABOUT Z-AXIS 45
1390 FORI=1TONP
1400 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1410 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1420 NEXT
```

Der folgende Programmteil legt über das Array E%() die Punkte der drei Würfel fest, die miteinander verbunden werden.

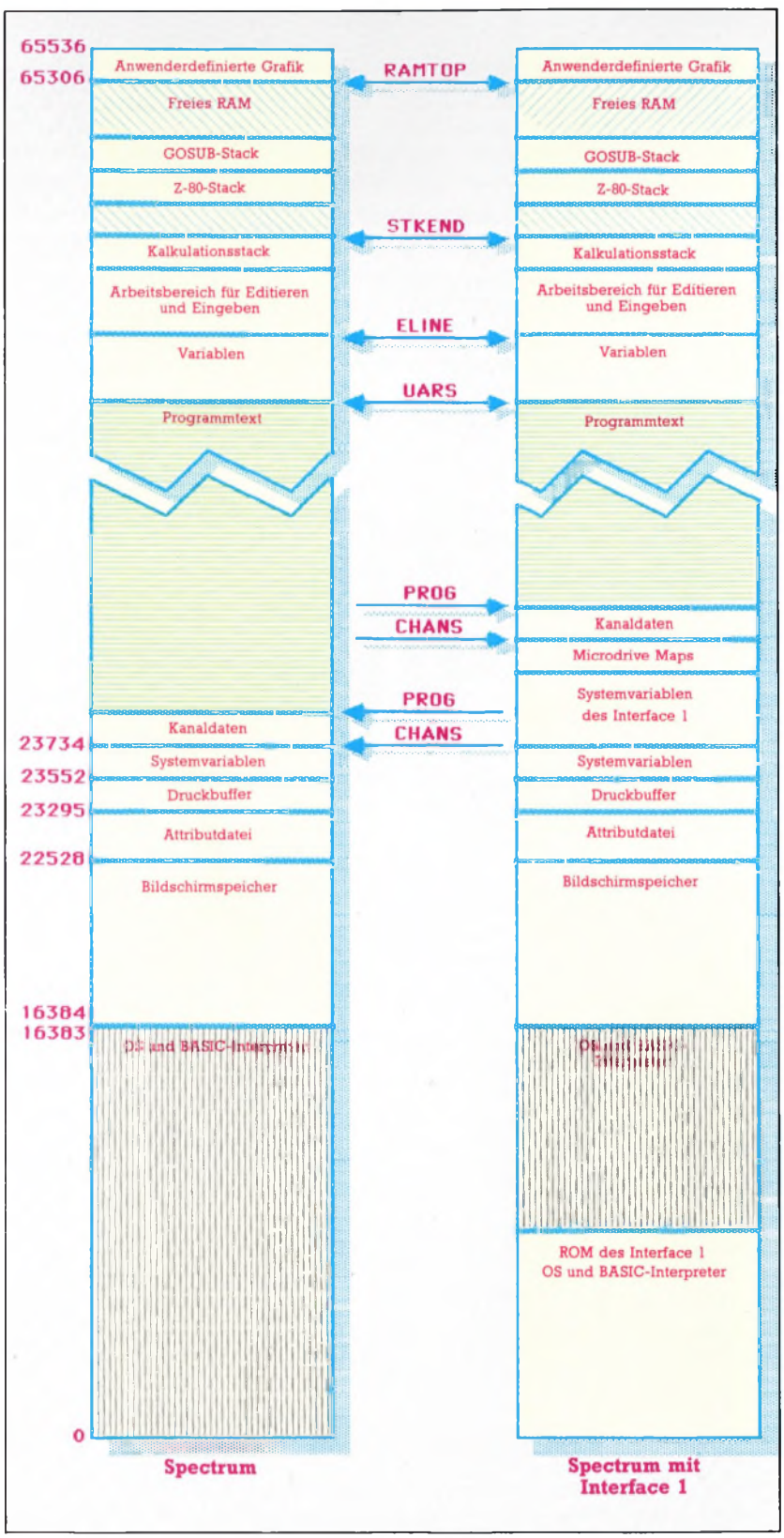
```
1430 REM**EDGE CONNECTION DATA**
1440 REM-MIDDLE BIG CUBE-
1450 E%(1,2)=1:REM 1 CONNECTED TO 2
1460 E%(2,3)=1:E%(3,4)=1:E%(4,1)=1
1470 E%(5,6)=1:REM BOT SQUARE
1480 E%(6,7)=1:E%(7,8)=1:E%(8,5)=1
1490 E%(5,1)=1:REM TOP TO BOT EDGES
1500 E%(6,2)=1:E%(7,3)=1:E%(8,4)=1
1510 REM-----------------
1520 REM-RIGHT SMALL CUBE
1530 FORI=9TO16:FORJ=9TO16
1540 E%(I,J)=E%(I-8,J-8)
1550 NEXT:NEXT
1560 REM-LEFT SMALL CUBE
1570 FORI=17TO24:FORJ=17TO24
1580 E%(I,J)=E%(I-8,J-8)
1590 NEXT:NEXT
1600 REM**SYMMETRISE E%(I,J)**
1610 FORI=1TONP:FORJ=1TONP
1620 IFE%(I,J)<>0THENE%(J,I)=1
1630 NEXT:NEXT
```

Das Programm ruft den Maschinencode auf, der die Rotationswerte berechnet und neue Figuren zeichnet. Die Maschinenroutine wird im Inneren der Schleife mehrfach aufgerufen, so daß sich die Figuren um 360 Grad drehen.

```
1640 REM##########################
1650 REM##PLOT ROTATING CUBE##
1660 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1670 GOSUB1790:REM INITIALISE
1680 FOR A=0 TO 2*π STEP SA
1690 SYS50536:REM ROTATE
1700 NEXTA:REM NEXT ANGLE
1710 GETA$:IFA$=""THEN1710
1720 REM##########################
1730 GOSUB1750:REM RESET SCREEN
1740 END
1750 REM**RESET SCREEN**
1760 POKE49408,0:SYS49422
1770 PRINTCHR$(147)
1780 RETURN
1790 REM**INITIALISE ROTSUB**
1800 REM************************
1810 REM**N.B. DO NOT DEFINE NEW**
1820 REM**VARIABLES BETWEEN THIS**
1830 REM**  SUBROUTINE & CALLING **
1840 REM**   MACHINE CODE ROTATE **
1850 REM** AS THIS WOULD CHANGE **
1860 REM**  BASE ADDRESS ARRAYS  **
1870 REM************************
1880 X(1)=X(1):REM X(1) CURRENT VAR
1890 POKE50500,PEEK(71):REM X(1) LO
1900 POKE50501,PEEK(72):REM X(1) HI
1910 Y(1)=Y(1):REM Y(1) CURRENT VAR
1920 POKE50502,PEEK(71):REM Y(1) LO
1930 POKE50503,PEEK(72):REM Y(1) HI
1940 Z(1)=Z(1):REM Z(1) CURRENT VAR
1950 POKE50504,PEEK(71)
1960 POKE50505,PEEK(72)
1970 POKE50506,NP:REM NUMBER OF POINTS
1980 CS=CS:REM MAKE CS CURRENT VAR
1990 POKE50507,PEEK(71)
2000 POKE50508,PEEK(72)
2010 SN=SN:REM MAKE SN CURRENT VAR
2020 POKE50509,PEEK(71)
2030 POKE50510,PEEK(72)
2040 E%(1,1)=E%(1,1):REM E% CURRENT VAR
2050 POKE50511,PEEK(71)
2060 POKE50512,PEEK(72)
2070 RETURN
```

Die vielen Talente des Spectrum

Die Memory Map des 48K Spectrum (und Spectrum+) zeigt deutlich, wie das Interface 1 die Belegung einiger RAM-Bereiche verändert. Die Systemvariablen erstrecken sich bis 23733.



Wir beginnen eine Serie über das Betriebssystem des Sinclair Spectrum und untersuchen zunächst die Memory Map des Computers. Dabei wird deutlich, welche Positionen sich für die Unterbringung von Maschinencode am besten und einfachsten eignen.

Der Sinclair Spectrum hat im Gegensatz zum Acorn B kein separates Betriebssystem-ROM. OS und BASIC-Interpreter des Spectrum (bzw. Spectrum+) befinden sich auf dem gleichen Chip. Das OS des Spectrum ist ein kompakter Satz von Routinen, der etwa halb soviel Codes umfaßt wie das OS des Acorn B. Es macht etwa sieben KByte der 16 KByte des ROM-Chips aus,

Trotz des langlebigen Betriebssystems und seines kompakten Codes hat der Spectrum im Vergleich zum Acorn B einen großen Nachteil. Da sein OS – von wenigen Ausnahmen abgesehen – nicht mit Vektoren arbeitet, ist eine Änderung bestimmter Abläufe recht kompliziert. Das Problem läßt sich jedoch umgehen. Wir kommen später darauf zurück.

In unserer Serie über das OS des Acorn B haben wir erfahren, daß das Betriebssystem die Routinen für Ein- und Ausgabe, Anzeige, Speicherung und Interrupts steuert und generell die Schnittstelle zwischen Hardware und dem Anwenderprogramm oder einem BASIC-Interpreter darstellt. Sehen wir uns zunächst die Memory Map des Spectrum im Bild an.

Den ersten Speicherblock von 16 KByte belegt das ROM mit BASIC-Interpreter und Betriebssystem. Das OS befindet sich in den unteren sechs bis sieben KBytes des Chips und der BASIC-Interpreter in der oberen Hälfte. Wenn das Gerät mit dem Interface 1 arbeitet, werden die unteren acht KByte des Speichers, ähnlich wie der Bereich von &8000 bis &BFFF auf dem Acorn B, durch „Paging" angesprochen. Beim Zugriff auf die Routinen des Interface 1, zum Beispiel zur Steuerung der Microdrives oder der seriellen Schnittstelle, wird das normale ROM ausgeblendet und statt dessen das ROM des Interface 1 angesprochen.

Den restlichen Speicher nimmt das RAM ein. Beim Einsatz des Interface 1 gibt es allerdings Unterschiede zum normalen Aufbau.

Der Bildschirmspeicher enthält die Daten der Abbildungen, die auf dem Bildschirm an-

gezeigt werden sollen. Die zugehörigen Farbdaten und Informationen wie BRIGHT und FLASH werden in der Attributdatei gespeichert, da sie sich jeweils auf ein Zeichenfeld der Anzeige beziehen. Der Druckbuffer besteht aus 256 Speicherbytes, die vom ZX-Drukker abgefragt werden, wenn der Befehl für eine Hardcopy eingegeben wird.

Die Systemvariablen werden vom OS und vom BASIC-Interpreter benutzt. Sie enthalten die wichtigsten Systemdaten. Eine vollständige Liste dieser Variablen ist im Anwenderhandbuch des Spectrum abgedruckt. Leider wurde sie nicht in das Handbuch des Spectrum+ übernommen.

Im Bereich der Systemvariablen treten die Unterschiede zwischen einem Spectrum, der mit oder ohne das Interface 1 arbeitet, am deutlichsten zutage. Das ROM des Interface 1 benötigt für seine Abläufe einen Arbeitsbereich und außerdem Systemvariablen, die es selbst erzeugt und hinter den normalen Systemvariablen des Spectrum ablegt. Da dabei jedoch die Aufteilung des restlichen Speichers ins „Rutschen" gerät, müssen einige Bereiche umorganisiert werden.

Auch die „Microdrive Maps" gibt es nur bei Verwendung des Interface 1. Sie geben an, welche Bereiche des Bandes belegt sind.

Bei den „Kanaldaten" hat man begrenzte Möglichkeiten, die Befehle LLIST, LPRINT, INPUT# und PRINT# über Vektoren einzusetzen. Sie können beispielsweise einen Maschinencodetreiber für einen Standarddrucker schreiben und die Kanaldaten so ändern, daß bei der Ausgabe von LLIST oder LPRINT Ihr Code und nicht die normalen Routinen des BASIC-Interpreters zum Einsatz kommen. In der nächsten Folge beschäftigen wir uns mit Abläufen dieser Art. Die Kanaldaten werden ebenfalls vom Interface 1 benötigt.

Editieren und Eingabe

Im Bereich Programmtext speichert das System den Text von BASIC-Programmen und deren Variablen.

Der Arbeitsbereich für „Editieren und Eingabe" wird vom System für das Editieren einzelner Programmzeilen oder die Eingabe von Daten und Programmzeilen eingesetzt. Da die Programmzeilen nach Beendigung des Editierens oder der Eingabe an den entsprechenden Positionen des Programms untergebracht werden, stehen diese Bereiche danach zur freien Verfügung.

Über diesem Arbeitsbereich liegen mehrere Stacks. Den „Kalkulationsstack" verwendet der BASIC-Interpreter für arithmetische Berechnungen. Mit dem Z80-Stack arbeitet die CPU auf ähnliche Weise, wie der 6502 die Speicherseite Eins benutzt: Dort werden die Rücksprungadressen der Maschinencodesubroutinen und zwischengespeicherte Daten untergebracht. Der GOSUB-Stack wird wiederum vom BASIC-Interpreter gebraucht. Er speichert die Rücksprungadressen der GOSUB-Befehle in BASIC. Schließlich gibt es noch einen Bereich für anwenderdefinierte Grafik mit den vom Anwender definierten Zeichen.

Die Memory Map zeigt deutlich, daß für selbstgeschriebene Maschinencodeprogramme kein Speicher reserviert wurde. Der Spectrum kann Maschinencode in drei Bereichen unterbringen: In einem REM-Befehl der Zeile 1. Die Speicherposition der ersten Zeile kann aus dem Wert der Systemvariable PROG errechnet werden. Wenn Sie auf PROG fünf addieren, erhalten Sie die Adresse des ersten Zeichens eines REM-Befehls in Zeile 1.

Maschinencode kann aber auch im Druckbuffer oder im Bereich zwischen RAMTOP und dem Speicher für anwenderdefinierte Zeichen abgelegt werden. Auf dem Acorn B ähnelt die letzte Methode dem Einsatz des Speichers zwischen HIMEM und dem Anfang des Video RAMs. Die folgende Tabelle zeigt die Vorteile der beiden Methoden:

Position	Spectrum	Mit Interface 1
REM in Zeile 1	✓	✗
Druckbuffer	(✓)*	(✓)*
Über RAMTOP	✓	✓

* Sollte bei Betrieb eines Druckers nicht verwandt werden

Es ist leicht zu erkennen, daß nur die letzte Möglichkeit in Frage kommt, da die beiden anderen in gewissen Situationen Probleme verursachen. Jeder Bereich oberhalb von RAM-TOP kann von BASIC nicht überschrieben werden. Er läßt sich außerdem leicht reservieren. Die genaue Position von RAMTOP ist in einer Zwei-Byte-Variablen bei den Adressen 23730 und 23731 gespeichert. Der Wert hat das übliche Z80-Format: Lo-Byte gefolgt Hi-Byte. Der Befehl liefert die aktuelle Position von RAMTOP:

```
LET ramtop=PEEK23730 + 256*PEEK23731
```

Nachdem Sie die Position von RAMTOP herausgefunden haben (auf einer standardmäßigen 48-K-Maschine liegt sie auf 65305), sollten Sie abschätzen, wieviele Byte Ihr Maschinencodeprogramm verlangt. Mit der Formel: Wert = RAMTOP − (Zahl-der-Programmbytes + 1) können Sie den neuen Wert errechnen und RAMTOP mit dem CLEAR-Befehl einfach heruntersetzen:

```
CLEAR Wert
```

Von einem BASIC-Programm aus löst dieser Befehl zunächst ein CLS aus, stellt dann die PLOT-Position auf den Ausgangswert zurück und ruft RESTORE auf. Außerdem wird der GOSUB-Stack gelöscht und verlegt. Sie sollten mit diesem Befehl vorsichtig umgehen und ihn möglichst am Programmanfang unterbringen.

Nach Ausführung von „CLEAR Wert" enthalten die Adressen 23730 und 23731 die neue Position von RAMTOP. Das erste Byte, das Ihnen für den Maschinencode zur Verfügung steht, befindet sich bei der Adresse (Wert + 1). Auf einer Standardmaschine mit 48 KByte reserviert der Befehl CLEAR 59999 etwa fünf KByte für Ihre Programme (Anfang bei 60000). Sie können Ihren Code mit POKE in den reservierten Bereich schreiben.

Keine OSBYTE und OSWORD

Sehen wir uns nun einige Systemvariablen an, die Informationen über die Speicheraufteilung enthalten. Zugriff auf die Variablen erhalten Sie mit PEEK oder POKE in die entsprechenden Speicherstellen. Es gibt jedoch keine Systemroutinen wie OSBYTE oder OSWORD, die auf dem Acorn B zur Verfügung standen.

- **chars**: Diese Variable liegt bei 23606 und 23607 und zeigt auf eine Speicheradresse, die sich 256 Bytes unter der Information über den Zeichensatz befindet. Normalerweise zeigt die in dieser Zwei-Byte-Variablen gespeicherte Adresse auf eine ROM-Adresse, die 256 Bytes unter dem ersten Byte der Definition des Leerzeichens (CHR$(32)) liegt. Die aktuelle Adresse der Variablen kann mit

```
LET chars=PEEK 23606 + 256*PEEK 23607
```

abgefragt werden. Wenn Sie einen eigenen, im RAM gespeicherten Zeichensatz verwenden wollen, brauchen Sie nur mit dem Wert dieser Adresse darauf zu zeigen.

- **vars**: Diese Variable (in 23627 und 23628) enthält die Anfangsadresse der gespeicherten BASIC-Variablen. Sie können diesen Wert zwar mit PEEK abfragen, sollten ihn aber nicht ändern, da der Spectrum dann seine Variablen nicht mehr „findet". Die Zeile

```
LET vars=PEEK 23627 + 256*PEEK 23628
```

ergibt die Anfangsadresse der Variablen.

- **prog**: Diese Systemvariable (bei 23635 und 23636) enthält die Anfangsadresse des BASIC-Programms und entspricht damit der Systemvariablen PAGE des ACORN B. Sie wird mit

```
LET prog=PEEK 23635 + 256*PEEK 23636
```

abgefragt. Der Befehl

```
PRINT vars-prog
```

zeigt die Länge des Programms an.

- **eline**: Die Variable ist an den Adressen 23641 und 23642 untergebracht und enthält die Anfangsadresse von Texten, die in das Gerät eingegeben werden. Sie können damit feststellen, wieviel Speicher die Variablen belegen. Wenn Sie vars bereits ermittelt haben, ergibt

```
LET eline=PEEK 23641 + 256*PEEK 23642
PRINT eline-vars
```

den von den Variablen belegten Speicher.

- **chans**: Diese Variable (bei 23631 und 23632) liefert die Anfangsadresse der Kanalinformation. Wir gehen in dieser Serie noch genauer auf die Kanäle ein.
- **stkend**: Diese Variable liegt an den Adressen 23653 und 23654. Zusammen mit dem Wert von RAMTOP gibt sie Auskunft darüber, wieviel Speicherplatz für BASIC-Programme und Variablen noch zur Verfügung steht.

```
LET stkend=PEEK 23653 + 256*PEEK 23654
```

errechnet den aktuellen Variablenwert und

```
PRINT ramtop — stkend
```

ergibt den verbleibenden freien Speicherplatz. Sie können den freien Speicher aber auch mit einer ROM-Routine errechnen, die jedoch nicht so präzise arbeitet wie die oben geschilderte Methode. Wenn Sie den aktuellen Wert von RAMTOP bereits haben, ergibt

```
PRINT ramtop — USR 7962
```

eine ungefähre Angabe über den noch freien Speicherplatz.

Assemblerzeit

Für die Besitzer eines Sinclair Spectrum, die mit Maschinencode arbeiten wollen, gibt es eine Reihe von Assemblerpaketen. Eines der bekanntesten ist DEVPAC 3 von Hisoft. Es wird auf Cassette geliefert und enthält den GENS3 Assembler und den Monitor/Disassembler MON3. Auch eine Microdrive-Version ist verfügbar. DEVPAC 3 ist zwar nicht besonders bedienerfreundlich, aber sehr preisgünstig.

Weiterhin gibt es die Doppelpakete Editor Assembler und Spectrum Monitor von Picturesque. Das System von Picturesque hat den Vorteil, daß es problemlos auf einem 16K Spectrum läuft. Die Handbücher dieser Pakete sind besser als die von Hisoft, und das gesamte System macht einen weit professionelleren Eindruck.

Alles mit Maßen

Dem Spectrum fehlt unter anderem eine komfortable Möglichkeit, die Länge eines Programms zu berechnen. Wir haben hier ein Dienstprogramm abgedruckt, das die Länge des im Speicher abgelegten BASIC-Programms mit den Systemvariablen PROG und VARS berechnet und in einem REM-Befehl der Zeile 1 ablegt. Das Modul stellt nur die Anforderung, daß die erste Zeile des Programms ein REM mit mindestens fünf Zeichen enthält:

```
1 REM 00000 Bytes lang
```

Die Ausführung des Maschinencodes schreibt in diese Zeile die Programmlänge, etwa:

```
1 REM 00801 Bytes lang
```

Das erste Listing enthält das Assemblerprogramm. Wenn Sie keinen geeigneten Assembler haben, können sie den Maschinencode auch einfach über das BASIC-Ladeprogramm einlesen.

In dem BASIC-Ladeprogramm setzt CLEAR 61999 den Wert der Systemvariablen RAMTOP auf 61999. Das erste „sichere" Byte befindet sich daher bei der Adresse 62000, und hier beginnt auch der Maschinencode. RAMTOP bleibt bis zur Ausführung eines weiteren CLEAR n oder eines PRINT USR 0 auf dem eingestellten Wert. Auch ein Ab- und Anschalten der Maschine löscht den Wert. Da NEW keinen Einfluß auf RAMTOP hat, kann der Maschinencode von BASIC nicht überschrieben werden, solange der geänderte RAMTOP aktiv ist.

Wenn sich das Programm im Speicher befindet, kann es mit USR aktiviert werden. Das Argument der Funktion USR(n) gibt die Adresse der Maschinenroutine an, die Sie aufführen wollen. Diese Adresse kann jede beliebige Speicheradresse sein. Die folgenden Befehle rufen daher alle einen Maschinencode bei Adresse 62000 auf und können unseren Programmcode aktivieren:

```
RANDOMISE USR(62000)
PRINT USER(62000/
LET L=USR(62000)
```

Das mit PRINT und LET eingesetzte USR liefert ein Ergebnis an BASIC zurück, das weiter verarbeitet werden kann. Das Resultat befindet sich beim Rücksprung des Maschinencodes zum BASIC in dem Registerpaar BC. Wenn Ihre Routine den Wert des BC-Registers nicht berührt, wird die Adresse der Routine zurückgeliefert.

Assemblerprogramm

```
62000                   10              ORG 62000
62000  2A4B5C   20              LD    HL,(23627)  ;get VARS in HL
62003  ED5B535C 30              LD    DE,(23635)  ;get PROG in DE
62007  AF       40              XOR   A           ;clear carry flag
62008  ED52     50              SBC   HL,DE       ;get program length into HL
62010  DD2A535C 60              LD    IX,(23635)  ;start of program into IX
62014  DD23     70              INC   IX
62016  DD23     80              INC   IX          ;increment IX to point to
62018  DD23     90              INC   IX          ;character after REM token
62020  DD23     100             INC   IX          ;in first program line
62022  DD23     110             INC   IX
62024  011027   120             LD    BC,10000    ;start getting ASCII codes
62027  CD6BF2   130             CALL  PRINT       ;of each digit in program
62030  DD23     140             INC   IX          ;length and store in REM
62032  01E803   150             LD    BC,1000     ;statement
62035  CD6BF2   160             CALL  PRINT
62038  DD23     170             INC   IX
62040  016400   180             LD    BC,100.
62043  CD6BF2   190             CALL  PRINT
62046  DD23     200             INC   IX
62048  010A00   210             LD    BC,10
62051  CD6BF2   220             CALL  PRINT
62054  DD23     230             INC   IX
62056  010100   240             LD    BC,1
62059  AF       250  PRINT:     XOR   A           ;clear carry and xero counter
62060  B7       260  L1:        OR    A
62061  ED42     270             SBC   HL,BC       ;subtract power of ten from
                                                  ;program length...
62063  3803     280             JR    C,FINISH    ;and jump if negative
62065  3C       290             INC   A           ;inc A reg if not negative
62066  18F8     300             JR    L1          ;repeat
62068  09       310  FINISH:    ADD   HL,BC       ;restore prog length to
                                                  ;positive value for the next
                                                  ;power of ten
62069  C630     320             ADD   A,48        ;get ASCII code of digit
62071  DD7700   330             LD    (IX),A      ;store ASCII code in REM
62074  C9       340             RET               ;finished
```

Ladeprogramm in BASIC

```
10 REM 00000 Bytes Long
20 CLEAR 61999
30 FOR i=0 TO 74
40 READ a: POKE 62000+i,a
50 NEXT i
1000 DATA 42,75,92,237,91,83,92,175,237,82,221,42,
83,92,221,35,221,35,221,35,221,35,221,35,1,16,39,2
05,107,242,221,35,1,232,3,205,107,242,221,35,1,100
,0
1010 DATA 205,107,242,221,35,1,10,0,205,107,242,22
1,35,1,1,00,175,183,237,66,56,3,60,24,-8,9,198,48,
221,119,0,201
```

Anmerkung: Um sicherzustellen, daß unser Programm die korrekte Position findet, an der die Ziffern mit der Programmlänge eingesetzt werden, sollten Sie zwischen der Zeilennummer und dem Schlüsselwort REM keine Leerzeichen einsetzen.

Kanalarbeiter

In unserer Serie über das Betriebssystem des Spectrum sehen wir uns die Steuerung der Kanäle an, über die Daten an den Bildschirm oder zum Drucker gesendet werden. Wir untersuchen dabei auch Systemvariablen, die in der Anzeige Veränderungen auslösen.

Strom	Kanal
0	K
1	K
2	S
3	P

Auf dem Sinclair Spectrum erfolgen Eingaben normalerweise per Tastatur, während Ausgaben hauptsächlich zum Bildschirm oder zum Drucker gehen. Im System des Spectrum wird jede dieser Hardwarekomponenten als Ein- oder Ausgabekanal bezeichnet. Daten, die in den Computer einfließen oder von dort kommen, heißen „Strom" (englisch: Stream). Datenströme lassen sich in unterschiedliche Hardwarekanäle lenken, wenn die angeschlossenen Geräte den Strom auf korrekte Weise verarbeiten können. Auf Ströme und Kanäle gehen wir jedoch erst in der nächsten Folge genauer ein.

In diesem Artikel untersuchen wir das E/A-System des Spectrum und sehen uns die Routine bei &0010 an, die ein Zeichen ausgibt. Jedes an diese Routine übergebene Zeichen wird je nach Kanalwahl an den Bildschirm oder den Drucker weitergeleitet. Unser Bild zeigt, wie auf einem nicht erweiterten Spectrum die Kanäle angelegt werden, und welche Nummern den entsprechenden Strömen zugeordnet sind. Die Kanäle werden mit einem einzelnen Buchstaben bezeichnet, die Ströme dagegen mit einer Nummer. Auf einem Standard-Spectrum sind nur die Ströme 0 bis 3 aktiv. Unsere kleine Tabelle am Rand zeigt, welche Kanäle welchen Stromnummern zugeordnet sind.

Um ein Zeichen an ein bestimmtes Gerät ausgeben zu können, müssen wir dem Spectrum zuerst mitteilen, welchen Strom er dafür aktivieren soll. Eine Ausgabe an den Bildschirm (Kanal S) geschieht über Strom 2. Mit einer ROM-Routine bei &1601 informieren wir das OS, welchen Strom es einsetzen soll. Vor Aufruf der Routine wird die Stromnummer im Register A gespeichert. Nach Ablauf der Routine ist der Hardwarekanal dieser Stromnummer eröffnet. Der Kanal S wird mit folgenden Befehlen eröffnet:

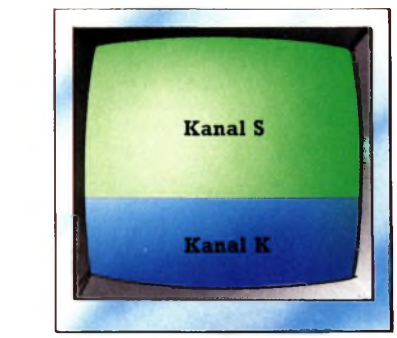
```
LD   A,2
CALL &1601
```

Nach der Öffnung eines Kanals wird der Code des Zeichens, das dorthin gesendet werden soll, in das A-Register gestellt und mit RST die Routine bei &0010 aufgerufen.

In einigen Aspekten entspricht dies dem OSWRCH-Aufruf des Acorn B. Wir sprechen hier die ROM-Adresse jedoch direkt an und nicht über einen Vektor. Kanal S arbeitet in dem Bildschirmbereich, der normalen PRINT-Befehlen in BASIC zugänglich ist. Er kann aber auch die beiden unteren Bildschirmzeilen anspre-

Kanalisierter Spectrum

S – Oberer Anzeigebereich
K – Ausgabe im unteren Anzeigebereich (von BASIC für Eingabe- und Fehlermeldungen eingesetzt) und Tastatureingaben.
P – Drucker



Auf dem Spectrum werden Daten in „Strömen" (die in Kanälen „fließen") zu Peripheriegeräten übertragen oder von dort empfangen. Das Einschalten des Geräts ordnet die Kanäle K, S und P den Strömen (0 und 1), 2 und 3 zu. PRINT #2 (Strom 2 – Bildschirm) entspricht dem normalen Befehl PRINT. Beispiel: PRINT #2; „Diese Meldung erscheint auf dem Bildschirm".

Kanal K Spectrum

Kanal P ZX Drucker

chen, die der BASIC-Interpreter normalerweise für Fehlermeldungen und Prompts einsetzt. Unsere Tabelle zeigt, daß diese Zeilen ein Teil von Kanal K sind.

Zeichen, die in diesen Zeilen erscheinen sollen, müssen über Kanal K gesendet werden:

```
LD   A,0
CALL &1601
```

Wenn das A mit 3 geladen wird, eröffnen die gleichen Befehle den Druckerkanal.

Außer Schriftzeichen lassen sich auch Steuerzeichen über einen Kanal senden. Die Wirkung der Steuerzeichen hängt natürlich von dem gerade aktiven Kanal ab. Das bedeutet, daß wir auch die Maschinencodeentsprechungen von PRINT AT, PRINT INK, PRINT PAPER etc. abrufen können. Die folgende Tabelle zeigt die Funktionen einiger Steuerzeichen für den Kanal S oder K. Einige der angegebenen Zeichen haben auf dem Drucker (Kanal P) natürlich keine Wirkung.

Code	Parameter	Wirkung
8	–	Cursor ein Zeichen nach links
9	–	Cursor ein Zeichen nach rechts
10	–	Cursor eine Zeile nach unten
11	–	Cursor eine Zeile nach oben
12	–	Löschen
13	–	ENTER
16	n	INK n (braucht ein zus. Byte)
17	n	PAPER n (braucht ein zus. Byte)
18	n	FLASH n (braucht ein zus. Byte)
19	n	BRIGHT n (braucht ein zus. Byte)
20	n	INVERSE n (braucht zus. Byte)
21	n	OVER n (braucht zus. Byte)
22	nn	AT y,x (braucht zwei zus. Bytes für die X- und Y-Koordinaten)
23	n	TAB n (braucht zus. Parameter)

Einige Steuerzeichen brauchen als Parameter zusätzliche Bytes, die normalerweise dem entsprechenden BASIC-Befehl folgen. Für die Ausführung von PAPER 3 werden beispielsweise die Bytes 17 und 3 zum Kanal S gesendet. Der folgende Maschinencode entspricht PRINT AT 10,10; „A":

```
3E02     10    ld   a,2      ;open channel S by
CD0116   20    call #1601    ;selecting stream 2
3E16     30    ld   a,22     ;token for AT
D7       40    rst  #10      ;output a char routine
3E0A     50    ld   a,10     ;y co-ordinate
D7       60    rst  #10
3E0A     70    ld   a,10     ;x co-ordinate
D7       80    rst  #10
3E41     90    ld   a,65     ;ASCII code for 'A'
D7      100    rst  #10
C9      110    ret
```

Beachten Sie, daß in der Liste das Steuerzeichen für CLS fehlt. Das Löschen des Bildschirms geschieht über eine Routine bei &06DB. Vor dem Aufruf der Routine muß Kanal S eröffnet werden. Nach Ablauf der Routine muß der Kanal nochmals eröffnet werden, damit Daten auf dem Bildschirm erscheinen können. Die folgende Routine löscht den Bildschirm:

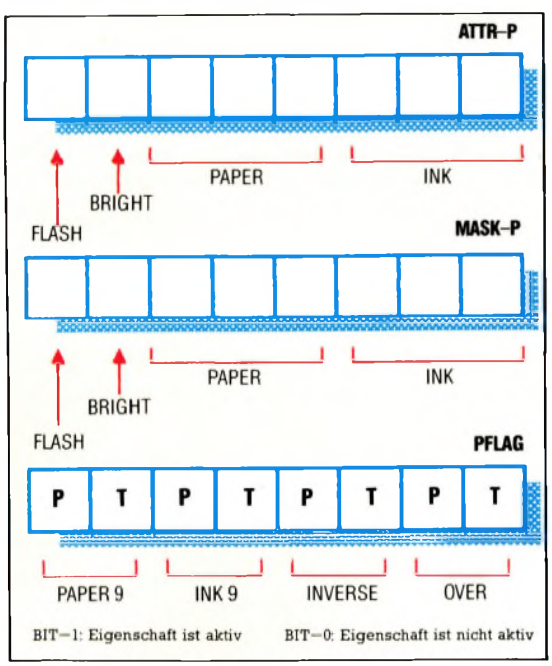
```
3E02     10    ld   a,2
CD0116   20    call #1601    ;open channel S
CDDB06   30    call #06db    ;clear the screen
3E02     40    ld   a,2
CD0116   50    call #1601    ;re-open channel S
C9       60    ret
```

Die Zeichenausgaberoutine hat die praktische Eigenschaft, daß dorthin übergebene Zahlen, die BASIC-Schlüsselwörter darstellen (Token), im Langtext dargestellt werden.

```
LD   A,249
RST &10
```

gibt daher auf Kanal S, K und P das Schlüsselwort RANDOMISE aus

Grafikbefehle wie PAPER, INK und BRIGHT, die mit der Zeichenoutputroutine ausgegeben werden, sind nur für die angegebene Folge von Ausgabezeichen aktiv – sie sind „temporäre Farbdaten". Wird der Befehl PAPER n außerhalb eines PRINT-Befehls ausgegeben, bleibt die entsprechende Farbe angeschaltet, bis ein weiterer PAPER-Befehl auftaucht.



Die beiden hierfür eingesetzten Systemvariablen heißen ATTR-P und MASK-P. Unsere Abbildung zeigt, wie sie verschiedene Aspekte der Anzeige steuern. Den drei Bits von ATTRP, die die Farbe von PAPER und INK festlegen, werden die in unserer Farbtabelle aufgeführten Werte zugeordnet. Wenn die Bits für FLASH oder BRIGHT auf Eins stehen, sind diese Eigenschaften aktiv. ATTR-P liegt bei der Adresse 23693.

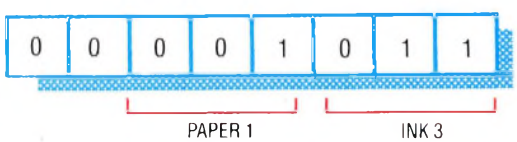
MASK-P befindet sich bei 23694. Jedes Bit (oder jede Bitgruppe) dieses Bytes, das auf Null steht, stellt sicher, daß der entsprechende Inhalt von ATTR-P die Bildschirmanzeige an dieser Position nicht verändert. Als praktisch erweist sich die Variable PFLAG bei 23697. Sie wird vom OS eingesetzt, um PAPER 9, INK 9,

Bits	Farbe
000	Schwarz
001	Blau
010	Rot
011	Magenta
100	Grün
101	Cyan
110	Gelb
111	Weiß

INVERSE und OVER anzuzeigen.

Zwei weitere Variablen – ATTR-T und MASK-T bei 23695 und 23696 – arbeiten ähnlich wie ATTR-P und MASK-P, steuern aber temporäre Farben (das heißt die Farbcodes, die mit PRINT-Befehlen oder Steuerzeichen über die Routine RTS &10 geschickt werden).

Der Einsatz von ATTR-P und MASK-P ist einfach. Für eine Farbänderung braucht man nur die entsprechenden Bits der Systemvariablen mit den Logikbefehlen des Maschinencodes AND und OR umzustellen. Für die Einstellung von PAPER 1:INK 3 sieht der Inhalt von ATTR-P so aus:



Geladen wird die Einstellung mit:

```
LD A,11
LD (23693),A
RET
```

Wie in BASIC haben die neuen PAPER- und INK-Befehle keine Auswirkung auf den bereits bestehenden Bildschirminhalt. Die neue PAPER-Farbe wird erst nach Ausführung eines CLS-Befehls aktiv.

Die Grafikroutinen des Spectrum sind Teil des BASIC-Interpreters. Beim Einsatz von PLOT und DRAW lassen sich durchaus auch Farbänderungen leicht mit den Variablen ATTR-P und MASK-P ausführen.

PLOT wird über Adresse &22E5 aufgerufen und erhält seine Koordinaten über das Registerpaar BC (B enthält die Y-Koordinate und C die X-Koordinate). Die Ausführung von PLOT 100,100 sieht im Maschinencode so aus:

```
LD B,100 ;Y-Koordinate
LD C,100 ;X-Koordinate
CALL &22E5 ;Ausführung
RET
```

Auch Farbänderungen lassen sich leicht durchführen. Die folgende Routine plottet einen roten Punkt auf den Bildschirm und versetzt ATTR-P vor seinem Rücksprung ins BASIC wieder in den ursprünglichen Zustand.

Kanal K arbeitet auf die gleiche Weise wie Kanal S. Alle Zeichen, die Sie in die unteren beiden Bildschirmzeilen schreiben, können jedoch von Meldungen des Betriebssystems oder des BASIC-Interpreters überschrieben werden.

Systemvariable LAST-K

Für Eingaben über diesen Kanal brauchen Sie nicht einmal eine ROM-Routine aufzurufen. Die Tastatur wird alle 20 Microsekunden abgefragt, wobei einige Systemvariablen anzeigen, ob eine Taste gedrückt wurde oder nicht. Die Systemvariable LAST-K (bei 23560) erhält den Zeichencode der zuletzt gedrückten Taste, während eine Systemvariable bei 23556 auf 255 steht, wenn keine Taste betätigt wurde. Die folgende Routine wartet auf einen Tastendruck (sie prüft, ob der Inhalt der Adresse 23556 nicht mehr auf 255 steht) und stellt den Code des eingegebenen Zeichens in das Register A. Nach dem Ablauf entspricht der Inhalt von LAST-K der zuletzt gedrückten Taste.

Weitere interessante Systemvariablen sind:

Variable	Adresse	Beschreibung
REPDEL	23561	Verzögerung der Tastenwiederholfunktion in Fünfzigstelsekunden. Legt Zeit bis zur Tastenwiederholung fest.
REPPER	23562	Verzögerung zwischen der zweiten und weiteren Wiederholungen – läßt sich mit POKE ansprechen.
PIP	23609	Länge des Tastaturklicks. Höhere Werte erzeugen ein „BEEP".

Der Kanal P des Standard-Spectrum spricht den Drucker über Strom 3 an. Die Befehle

```
LD   A,3
CALL &1601
```

eröffnen Kanal P. Die darauf folgenden Zeichen werden auf dem Drucker ausgegeben. BASIC arbeitet oft mit Kanälen: PRINT und LIST sprechen Kanal S an, LPRINT und LLIST Kanal P und INPUT Kanal K. Man kann jedoch auch eigene Kanäle definieren, die dann auf Zusatzgeräte wie Microdrives, weitere Drucker und andere Hardware zugreifen. In der nächsten Folge gehen wir ausführlich darauf ein.

```
3A8D5C    10        ld   a,(23693)      ;get ATTR-P in A reg
F5        20        push af             ;and save on stack
E6F8      30        and  248            ;clear low 3 bits to 0
F602      40        or   2              ;set bit 1 to give red ink
0664      50        ld   b,100          ;y co-ordinate
0E64      60        ld   c,100          ;x co-ordinate
CDE522    70        call #22e5          ;call PLOT
F1        80        pop  af             ;restore original
328D5C    90        ld   (23693),a      ;ATTR-P contents
C9        100       ret
```

```
          10 ; GET-CHAR routine
3A845C    20 key:   ld   a,(23556)      ;check to see if
FEFF      30        cp   255            ;key is being pressed
28F9      40        jr   z,key          ;keep checking
3A885C    50        ld   a,(23560)      ;get LAST-K in A
C9        60        ret
```

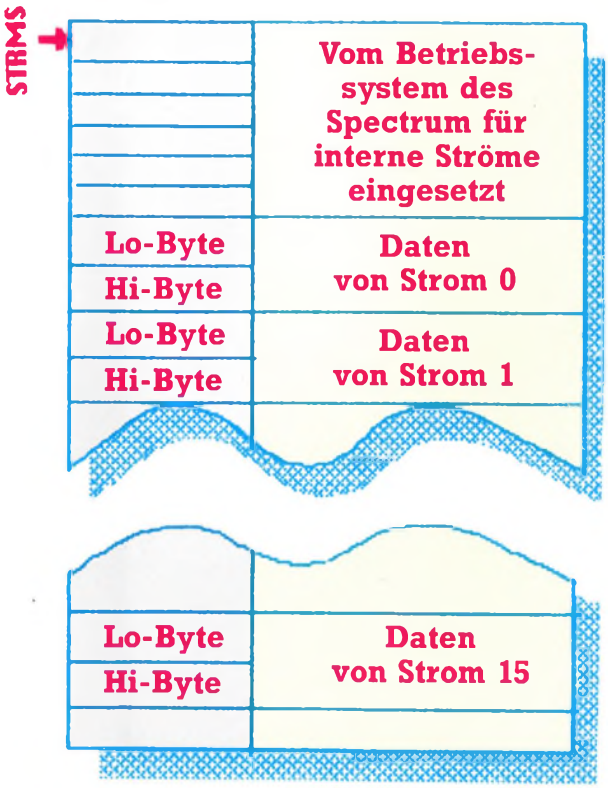
Stromlinien

In der vorigen Folge hatten wir gezeigt, wie das Betriebssystem des Spectrum seine Ein- und Ausgabekanäle anwählt und steuert. Nun betrachten wir die Datenströme, die über diese Kanäle „fließen".

Die Initialisierung des Spectrum ordnet beim Anschalten oder bei einem Reset Datenströme bestimmten Kanälen zu. Unsere Tabelle zeigt diese Zuordnungen:

Kanal	Strom	Kanalbeschreibung
K	0	Tastatur und unterer
K	1	Bildschirmbereich
S	2	oberer Bildschirmbereich
P	3	ZX-Drucker

Es gibt jedoch auch Ströme, die auf einem Standard-Spectrum nicht genutzt werden. Beachten Sie daher, daß sich die folgenden Beschreibungen auf einen Spectrum ohne Interface 1 beziehen. Wenn das lokale Netzwerk, die Microdrives oder die serielle Schnittstelle in Betrieb sind, ändert sich der gesamte Aufbau. Wir gehen auf die Ausbaustufen des Spectrum zu einem späteren Zeitpunkt genauer ein. Die weiteren Ströme haben die Nummern 4 bis 15 und können jedem E/A-Kanal zugeordnet werden, den Sie als Schnittstelle einsetzen wollen.



Das Betriebssystem des Spectrum speichert die Information, welche Ströme welchen Kanälen zugeordnet sind, in den 38 Bytes der Tabelle STRMS. Die Tabelle befindet sich im Bereich für Systemvariablen bei 23568. Die ersten sechs Tabellenbytes braucht das OS für eigene Zwecke.

Die weiteren Bytes sind paarweise jedem der 16 Ströme zugeordnet. Unser Bild zeigt den Tabellenaufbau.

Inhalte der Tabelle

Das folgende BASIC-Programm stellt den Inhalt der Tabelle dar:

```
10 FOR I=23568 TO 23604 STEP 2
20 PRINT PEEK I + 256 * PEEK (I+1)
30 NEXT I
```

Die ersten drei Einträge bezeichnen die internen Ströme des Spectrum. Alle darauffolgenden Werte sind Vektoren der Ströme 0 bis 15 und zeigen auf die Einträge einer weiteren Tabelle. Die Systemvariable CHANS (bei 23631) gibt an, wo diese Tabelle liegt. Man kann die Adresse mit

```
PRINT PEEK 23631 + 256 * PEEK 23632
```

herausfinden. Die Systemvariable CHANS sollte nie verändert werden, da sie auf den Anfang der Tabelle für Kanaldaten zeigt, die wichtige Informationen über jeden Kanal enthält. Der Wert der STRMS-Tabelle dient als Index für die Kanaldatentabelle.

Der entsprechende Eintrag gibt mit (Offset + 1) an, wo sich – vom Tabellenanfang aus – die Informationen über einen bestimmten Kanal befinden. Der STRMS-Tabelleneintrag für Strom 3 (Drucker) ist daher 16. Wenn Sie davon Eins abziehen, erhalten Sie den Offset von 15. Die Daten über Kanal 3 befinden sich also bei der Adresse

```
(PEEK 23361 + 256 * PEEK 23632) + 15
```

(vorausgesetzt, die Strom- und Kanalzuordnung wurde nach der Initialisierung nicht verändert).

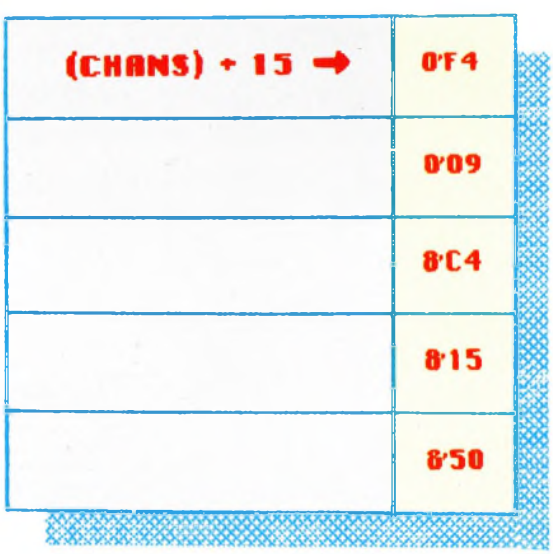
Ein Null-Eintrag der STRMS-Tabelle zeigt an, daß der Strom noch keinem Kanal zugeordnet wurde. Auf einem Standard-Spectrum ist das bei Strom 4 bis 15 der Fall.

Jeder Eintrag der Kanaldatentabelle hat das Format:

Lo-Byte	Adresse der Ausgaberoutine
Hi-Byte	des Kanals
Lo-Byte	Adresse der Eingaberoutine
Hi-Byte	des Kanals
n	Einzelnes Byte, das den Buchstaben des Kanalcodes im ASCII-Format darstellt, d. h. K, S oder P

Ein Eintrag der STRMS-Tabelle zeigt auf das zweite Byte des entsprechenden Eintrags in der Kanaldatentabelle. Daher besteht die Notwendigkeit, Eins zu subtrahieren. Diese Tabelle bildet das Herz des E/A-Systems. Wenn ein Kanal einen E/A-Vorgang nicht ausführen kann, zeigt der entsprechende Eintrag auf eine ROM-Routine, die die Fehlermeldung „Invalid I/O Device" ausgibt. Auf den reinen Ausgabekanälen (etwa Kanal S) zeigt der Adreßeintrag der Eingaberoutine auf dieses Fehlermodul. In jedem anderen Fall steht dafür die entsprechende ROM-Routine für Ein- oder Ausgabe zur Verfügung.

Sehen wir uns den Eintrag für Kanal P einmal genauer an. Normalerweise ist er ein reiner Ausgabekanal, der Daten zum ZX-Drucker sendet. Seine Tabellendaten haben wir grafisch dargestellt.



Zeigt auf die GP Druckroutine bei Adresse 09F4 im ROM

Zeigt auf die Routine zur Erzeugung von Fehlermeldungen bei Adresse 15C4 im ROM

ASCII-Code für „P"

Der Eintrag für Kanal S liegt bei (CHANS)+5 und der von Kanal K bei (CHANS)+0. (CHANS) ist die in der Systemvariable CHANS gespeicherte Adresse.

Mit den Befehlen OPEN# und CLOSE# lassen sich nicht belegte Ströme den Kanälen zuordnen. Sie kommen jedoch erst in einem erweiterten System (z. B. mit Interface 1) voll zum Tragen. Hier das Einsatzformat dieser Befehle auf einem Standard-Spectrum:

```
OPEN #4,"S"
```

ordnet Strom 4 dem Kanal S zu. Wenn Sie diesen Befehl ausgeben und dann die STRMS Tabelle untersuchen, werden Sie unter Strom 4 einen Eintrag finden. Der Befehl kann auch Variablen verarbeiten:

```
10 LET n=4
20 LET c$="S"
30 OPEN #n,c$
```

führt exakt das gleiche aus, wie der einzeilige OPEN-Befehl. Nach Öffnen eines Kanals können Sie mit PRINT# und LIST# Daten über diesen neuen Strom senden.

```
40 PRINT#4;"Hallo"
```

gibt „Hallo" auf dem Bildschirm aus. LIST #4 listet dort das aktuelle Programm. Wenn ein Strom einem Kanal zugeordnet wurde, der sich für Ein- und Ausgabe eignet, kann auch der Befehl INPUT# eingesetzt werden. Die erweiterten PRINT- und INPUT-Anweisungen zeigen ihren Nutzen jedoch nur dann, wenn die zusätzlichen Kanäle des Interface 1 zur Verfügung stehen.

CLOSE und OPEN

Der Befehl CLOSE# löst Kanal- und Stromzuordnungen auf. Die Aussage von CLOSE#4 (nach dem obenstehenden OPEN-Befehl) koppelt daher Strom 4 von Kanal S ab. Danach erzeugt jeder Versuch, Daten in den „geschlossenen" Strom zu senden, eine Fehlermeldung.

Die Befehle PRINT#, LIST#, INPUT# und INKEY$# lassen sich auch mit Strom 0 bis 3 einsetzen. Diese Ströme sind ständig eröffnet, so daß der Befehl

```
10 CLOSE #2:REM sollte Kanal S schließen
20 PRINT #2; "Hallo"
```

keine Fehlermeldung erzeugt: Das OS stellt sicher, daß der Kanal wieder eröffnet ist, wenn Daten dorthin gesandt werden. Was löst nun der Befehl LIST# aus?

```
LIST#3
```

führt unter normalen Umständen den gleichen Ablauf aus wie LLIST – das Programm wird auf Kanal P (Drucker) ausgegeben. LIST#0 und PRINT#0 sind schon interessanter, da sie den unteren Bildschirmbereich ansprechen (der für normale PRINT-Befehle nicht erreichbar ist). Denken Sie jedoch daran, daß dieser Bereich von dem OS des Spectrum mit Meldungen übrschrieben wird. PRINT#3 entspricht unter normalen Umständen LPRINT.

Das Verhalten eines Kanals kann auch verändert werden. So wird Kanal P oft umgestellt, um über eine entsprechende Schnittstelle einen Centronics Drucker betreiben zu können. Wenn Sie in der Kanaldatentabelle den Eintrag für die Ausgaberoutine ändern, dann senden LPRINT und LLIST die Daten an eine Routine, die einen Centronics-Drucker steuert. Statt des Druckers lassen sich auch andere Geräte mit der Schnittstelle verbinden.

Hi-Res Printprogramm

Das folgende Programm leitet Daten, die über den Kanal P des Spectrum gesandt werden, zu einer Routine, die sie in hoher Auflösung anzeigt. Das Listing kann über einen Standardassembler eingegeben werden. Sie können den Hexcode auch direkt mit dem Monitor oder dem ebenfalls abgedruckten Hex-Ladeprogramm eingeben. Nach Eingabe und Aufruf des Hex-Ladeprogramms werden Sie nach der Anfangsadresse gefragt (geben Sie hier 65068 an). Sie können nun den Hexcode zweizeichenweise eingeben. Die Codezeile etwa

2130FE

wird dabei als

21 (ENTER)
30 (ENTER)
FE (ENTER)

eingetippt. Sie beenden das Programm, indem Sie S eintippen. Über die Prüfsumme (29155) können Sie nun feststellen, ob die Eingabe wirklich fehlerfrei ist.

Der neue, vom Programm gesteuerte Befehl hat das Format:

LRPINT AT, X,Y;
"MESSAGE"

wobei X zwischen 0 und 168 und Y im Bereich von 0 bis 248 liegen muß. Die „MESSAGE" kann auch eine Stringvariable oder eine Zahl sein. Falls der mit dem neuen Befehl gedruckte String über den Bildschirmrand hinausgeht, setzt er sich zwar am linken Bildschirmrand fort, jedoch auf der gleichen Zeile.

Hex-Ladeprogramm

```
  10 CLEAR 63999
  15 POKE 23658,8
  17 DEF FN H(H$)=16*(CODE (H$)-48-7*(H$
((1)>"9"))+(CODE (H$(2))-48-7*(H$(2)>"9")
)
  20 INPUT "Start address ";a
  25 LET Z=A
  30 INPUT (a), LINE a$
  35 IF A$="S" THEN  GO TO 90
  40 PRINT a,a$: POKE 23692,255
  50 FOR Z=1 TO LEN A$/2
  60 POKE a,FN H(a$(((z*2)-1) TO )): LET
 a=a+1
  70 NEXT z
  80 GO TO 30
  90 INPUT "CHECKSUM:";D
 100 LET C=0: FOR B=Z TO A-1: LET C=C+PE
EK B: NEXT B
 110 IF C=D THEN  PRINT "The code is OK!
": STOP
 120 PRINT "Oh dear , the code is incorr
ect": STOP
```

Assemblerlisting

```
                ORG   65068
          PIXAD EQU   22AAH
          UDG   EQU   23675
          CHARS EQU   23606
                ;     "Make the data in CHANS point to our routine"
2A4F5C    ENABL LD    HL,(23631)
010F00          LD    BC,15
09              ADD   HL,BC
01?AFE          LD    BC,DO-IT
71              LD    (HL),C
23              INC   HL
70              LD    (HL),B
C9              RET
                ;     "Preserve regesters and call new print routine"
E5        DO-IT PUSH  HL
C5              PUSH  BC
D5              PUSH  DE
F5              PUSH  AF
CD46FE          CALL  DOIT1
F1              POP   AF
D1              POP   DE
C1              POP   BC
E1              POP   HL
C9              RET
                ;     "Check to see if a holdsan AT control code"
F5        DOIT1 PUSH  AF
3A1DFF          LD    A,(ATFLG)
FE00            CP    0
200B            JR    NZ,GETXP
F1              POP   AF
FE16      ATCHQ CP    22
201D            JR    NZ,CRCHQ
3EFF            LD    A,255
321DFF          LD    (ATFLG),A
C9              RET
FEFE      GETXP CP    254
2809            JR    Z,GETYP
F1              POP   AF
3216FF          LD    (XPOSI),A
211DFF          LD    HL,ATFLG
35              DEC   (HL)
C9              RET
F1        GETYP POP   AF
3217FF          LD    (YPOSI),A
3E00            LD    A,0
321DFF          LD    (ATFLG),A
C9              RET
                ;     "Check for end of print data (signified by 13)"
FE0D      CRCHQ CP    13
2001            JR    NZ,VCHRQ
C9        SKIPC RET
                ;     "If the code in a is    between 32 & 128 or if"
                ;     "it is a UDG code then  find its data in memory"
FE20      VCHRQ CP    32
380C            JR    C,PRNT?
FE80            CP    128
381C            JR    C,FCHR
FE90      UDGCQ CP    144
3804            JR    C,PRNT?
FEA5            CP    165
3804            JR    C,FUDGC
                ;     "For all other codes a ?will be printed"
3E3F      PRNT? LD    A,63
1810            JR    FCHR
                ;     "Find the data for the  UDG or character in mem"
D690      FUDGC SUB   144
210000          LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A7B5C          LD    HL,(UDG)
19              ADD   HL,DE
180C            JR    PRNIT
210000    FCHR  LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A365C          LD    HL,(CHARS)
19              ADD   HL,DE
010700    PRNIT LD    BC,7
09              ADD   HL,BC
221BFF          LD    (CHRAD),HL
                ;     "Check to see that the X& Y positions are valid"
3A16FF          LD    A,(XPOSI)
FEF9            CP    249
D231FF          JP    NC,ERRB
3A17FF          LD    A,(YPOSI)
FEA9            CP    169
3078            JR    NC,ERR5
ED4B16FF        LD    BC,(XPOSI)
CDAA22          CALL  PIXAD
321AFF          LD    (PIXPO),A
221BFF          LD    (DFADD),HL
                ;     "Get the first pixel slice of char being printed"
0608            LD    B,8
C5        PRNLP PUSH  BC
2A1BFF          LD    HL,(CHRAD)
7E              LD    A,(HL)
2B              DEC   HL
221BFF          LD    (CHRAD),HL
6F              LD    L,A
                ;     "If the pixel slice doesn't need moving within"
                ;     "the display byte then  jump forward"
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAE6FE          JP    Z,PUTIT
                ;     "Move the slice along in the display bytes to"
                ;     "the correct pixel position"
47              LD    B,A
2600            LD    H,0
CB3D      ROTLP SRL   L
CB1C            RR    H
A7              AND   A
10F9            DJNZ  ROTLP
                ;     "Put the slice at the   right screen location"
ED5B1BFF  PUTIT LD    DE,(DFADD)
1A              LD    A,(DE)
AD              XOR   L
12              LD    (DE),A
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAFAFE          JP    Z,PST
13              INC   DE
1A              LD    A,(DE)
AC              XOR   H
12              LD    (DE),A
1B              DEC   DE
2A1BFF    PST   LD    HL,(DFADD)
CD1EFF          CALL  ULINE
221BFF          LD    (DFADD),HL
C1              POP   BC
10C4            DJNZ  PRNLP
                ;     "Add 8 to the X positionso that the next char"
                ;     "is printed in a clear  space"
3A16FF          LD    A,(XPOSI)
C608            ADD   8
                ;     "If the end of the line is reached go back to"
                ;     "the start BUT DO NOT   move down a line"
FEF9            CP    249
DA12FF          JP    C,STAIT
3E00            LD    A,0
3216FF    STAIT LD    (XPOSI),A
C9              RET
00        XPOSI DEFB  0
00        YPOSI DEFB  0
0000      DFADD DEFW  0
00        PIXPO DEFB  0
0000      CHRAD DEFW  0
00        ATFLG DEFB  0
                ;     "Move the address in HL one pixel up the screen"
F5        ULINE PUSH  AF
7C              LD    A,H
25              DEC   H
E607            AND   7
200A            JR    NZ,END
7D              LD    A,L
D620            SUB   32
6F              LD    L,A
3804            JR    C,END
7C              LD    A,H
C608            ADD   8
67              LD    H,A
F1        END   POP   AF
C9              RET
                ;     "Create various errors"
CF        ERRB  RST   8
0A              DEFB  10
CF        ERR5  RST   8
04              DEFB  4
          FINIS END
```

Bandmaß

Wir untersuchen, wie das Betriebssystem des Spectrum die Cassetten mit Hilfe von ROM-Routinen anspricht und zeigen, wie sich diese Routinen auch vom Maschinencode aus einsetzen lassen.

Der Standard Sinclair Spectrum besitzt nur eine Cassettenrecorder-Schnittstelle, über die alle Programme und Daten geladen und gesichert werden. Erst der Einbau eines Interface 1 eröffnet den Zugang zu Microdrives, serieller Datenübertragung und lokalen Netzwerken (die – wie die alternativen Kanäle des Acorn B – als zusätzliche Dateisysteme angesehen werden). Während der Acorn B seine Dateisysteme mit dem *-Befehl anwählt, arbeitet der Spectrum jedoch mit einer völlig andersartigen Syntax, um zwischen Befehlen, die für das Cassettensystem gedacht sind und alternativen Dateisystemen unterscheiden zu können. So sichert beispielsweise der Befehl

SAVE "fred1" LINE 1

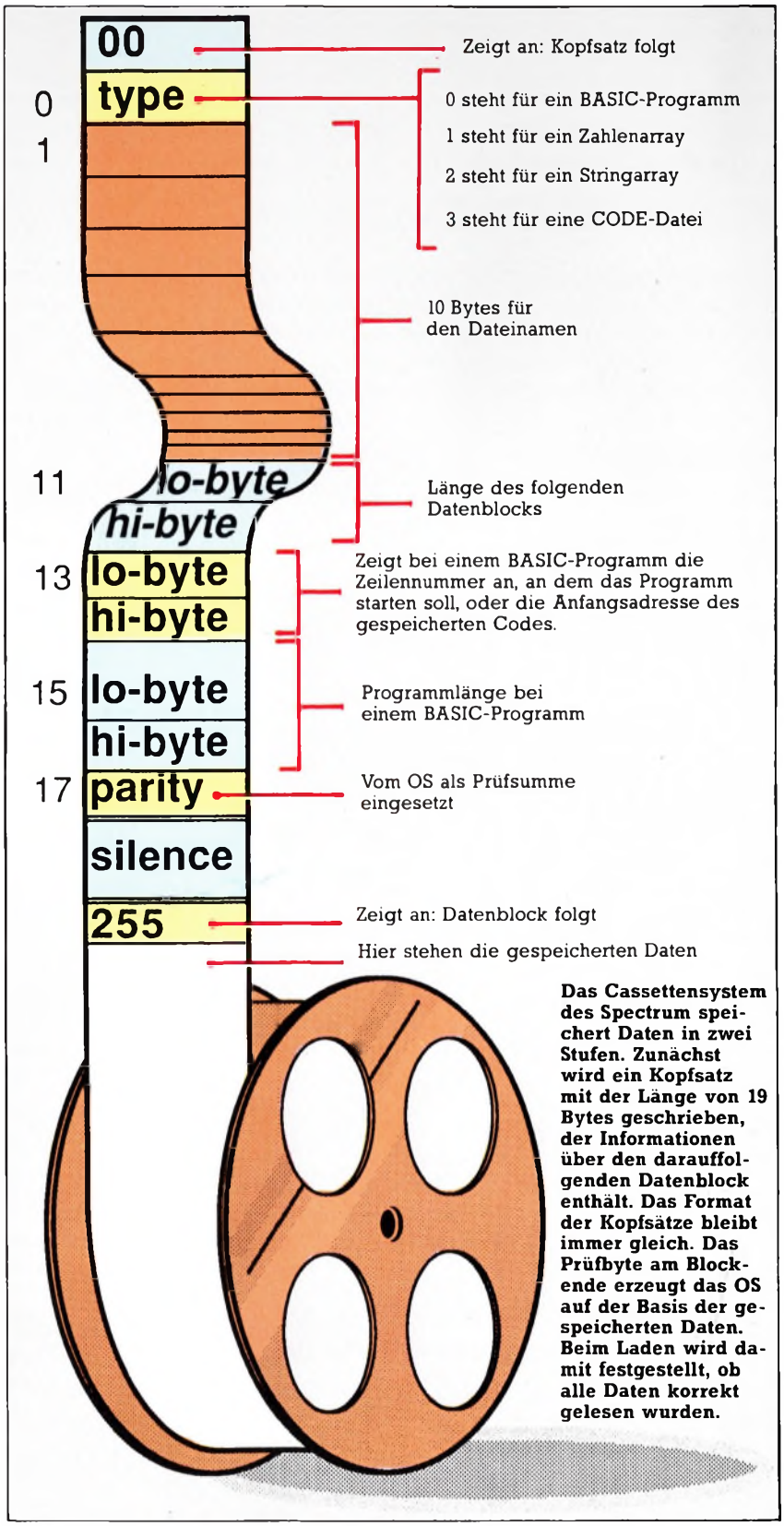
ein BASIC-Programm auf Cassette, das beim Zurückladen automatisch auf Leitung 1 ausgeführt wird. Der gleiche Befehl für Microdrives ist weit umständlicher:

SAVE *"m";1;"fred1" LINE 1

Das „m" gibt an, daß Microdrives angesprochen werden, während die darauffolgende 1 die Nummer des Laufwerks festlegt. Mit der Funktionsweise des Microdrivesystems wollen wir uns jedoch erst später beschäftigen. Zunächst wird demonstriert, wie der Maschinencode das Cassettensystem steuern kann.

Leerzeichen füllen auf

Alle Cassettendaten werden auf die gleiche Art gespeichert – unabhängig von ihrer Struktur (siehe Bild). Die ersten 19 Bytes bilden den Kopfsatz mit Informationen über den darauffolgenden Datenblock. Die jeweils ersten und letzten Bytes des Kopfsatzes werden von Betriebssystemroutinen definiert und auf das Band geschrieben. Die anderen Bytes müssen vor Aufruf des Schreibmoduls definiert werden. Beim Laden zeigt das „Typbyte" dem OS an, welche Art Daten der Block enthält. Darauf folgen zehn Bytes mit dem Dateinamen. Enthält ein Dateiname weniger als zehn Zeichen, werden die restlichen Positionen mit Leerzeichen (ASCII-Code 32) gefüllt. Von Byte 11 an liefert der Kopfsatz dem OS weitere Informationen – beispielsweise in welchen Speicherbereich die Daten geladen werden sollen. Das letzte



Das Cassettensystem des Spectrum speichert Daten in zwei Stufen. Zunächst wird ein Kopfsatz mit der Länge von 19 Bytes geschrieben, der Informationen über den darauffolgenden Datenblock enthält. Das Format der Kopfsätze bleibt immer gleich. Das Prüfbyte am Blockende erzeugt das OS auf der Basis der gespeicherten Daten. Beim Laden wird damit festgestellt, ob alle Daten korrekt gelesen wurden.

Byte des Kopfsatzes ist ein Prüfbyte, mit dem getestet wird, ob beim Lesen Fehler aufgetreten sind. In diesem Fall erscheint eine entsprechende Meldung auf dem Bildschirm.

Am Anfang des Hauptdatenblockes steht immer ein Byte mit dem Wert 255, das eben-

falls vom Schreibmodul eingesetzt wird. Danach kommen Daten, die mit einem weiteren, vom Schreibmodul erzeugten Prüfbyte abschließen. Der Inhalt der Prüfbytes hängt von den Daten des davorstehenden Blockes ab.

Wie werden die Cassettenroutinen nun von Maschinencodeprogrammen eingesetzt? Die Routinen für die Cassettensteuerung liegen im ROM des Spectrum zwischen &04C2 und &09F3. Sie sind recht umfangreich, da die Zeitabläufe und die Tonerzeugung für das Beschreiben des Bandes per Software erzeugt werden.

Zwar gibt es keine speziellen Systemvariablen für Cassettenfunktionen, doch zeigt die folgende Tabelle einige Variablen, die für diese Aufgabe eingesetzt werden.

Systemvariable	Beschreibung
TADDR (bei 23668 und 23669)	Entscheidet bei der Interpretation eines BASIC-Befehls, welcher Cassettenvorgang von den ROM-Routinen ausgeführt werden soll.
BORDCR (bei 23624)	Stellt nach einem Cassettenvorgang die Hintergrundfarbe wieder her.
XPTR (bei 23647 und 23648)	Zwischenspeicher für das Register IX

Auch die Systemvariablen VARS, ELINE und PROG werden vom Spectrum für die Cassettenspeicherung verwandt.

Die Einsprungbedingungen

Sehen wir uns zunächst an, wie Daten auf Band gespeichert werden (die Routine liegt bei &04C2). Im Normalfall wird diese Routine zweimal aufgerufen – einmal zur Speicherung des Kopfsatzes und ein zweites Mal zum Schreiben des Datenblocks. Unsere zweite Tabelle gibt die Einsprungbedingungen an.

Register	Kopfsatz	Datenblock
A	0	255
DE	17	Länge der Daten
IX	Anfangsadresse des Kopfsatzes oder des Datenblocks	

Dabei zeigt das IX-Register auf die Daten, die geschrieben werden sollen. In dem folgenden Programmbeispiel schreiben wir einen Datenblock von 100 Bytes (Anfang bei ROM-Adresse 0000) auf die Cassette.

```
        ;routine to save 100 bytes, starting address
         0000, to tape
        ;
        ;send header
3E00            ld   a,0        ;indicate a HEADER BLOCK
DDE5            push ix         ;save ix on stack
DD2100          ld   ix,header  ;add. of header block
111100          ld   de,17      ;no of header bytes
CDC204          call #04c2      ;write header to tape
        ;send data
DD210000        ld   ix,0000    ;add of 1st byte to save
116400          ld   de,100     ;no of bytes to be saved
3EFF            ld   a,255      ;indicate a DATA BLOCK
CDC204          call #04c2      ;write data to tape
DDE1            pop  ix         ;restore IX value
C9              ret             ;back to BASIC
03      header: defb 3          ;data type 3=CODE block
54455354        defm "TESTPROG" ;fname+spaces=10 chars
64              defb 100        ;lo-byte of data-length
00              defb 0          ;hi-byte of data-length
00              defb 00         ;lo-byte of start add.
00              defb 00         ;hi-byte of start add.
00              defb 00         ;used for BASIC only
00              defb 00         ;for start line no.
```

Der Aufruf dieser Routine sichert den angegebenen Speicherbereich. Im Gegensatz zu den BASIC-Bandroutinen gibt diese Routine keine Meldung auf dem Schirm aus. Da die Banddaten verschiebbar sind, bietet die Verifizierung keine Schwierigkeiten.

Laden Sie mit dem Befehl

```
LOAD "TESTPROG" CODE 40000
```

das Programm wieder, und vergleichen Sie die geladenen Bytes mit den Speicherstellen des ROM von 0 bis 99. Bedenken Sie aber, daß der Kopfsatz zwar mit dem Typbyte anfängt, das erste auf Band geschriebene Byte jedoch vom Betriebssystem geliefert wird (0 für Kopfsatz und 255 für Datenblock).

Durch eine kleine Veränderung lassen sich auch BASIC-Programme vom Maschinencode aus sichern. Der Kopfsatz am Ende des vorigen Listings muß dann lauten:

```
00      header: defb 0          ;0=BASIC program
73737373        defm "sssss     ;fname filled to 10 chars
                sssss"
00              defb nn         ;lo-byte) length of
00              defb nn         ;hi-byte) prog+variables
00              defb nn         ;lo-byte) start line
00              defb nn         ;hi-byte) number
00              defb nn         ;lo-byte) length of
00              defb nn         ;hi-byte) program only
```

Die Länge von Programm und Variablen erhalten Sie, wenn Sie den Wert PROG von ELINE abziehen. Die Programmlänge ergibt sich, wenn Sie PROG von VARS subtrahieren. Falls das Programm nicht sofort nach dem Laden starten soll, brauchen Sie im Listing nur die Zeilennummer des Programmanfangs auf 32768 zu setzen. Auf ähnliche Weise kann man auch den Befehl SAVE SCREEN$ vom Maschinencode aus simulieren. Zu diesem Zweck muß man die Anfangsadresse des Codes, der gesichert werden soll, auf %4000 setzen und die Länge mit &1B00 angeben.

Mit den Techniken, die wir früher im Kurs behandelt haben, läßt sich die Routine leicht ausbauen. So können Sie beispielsweise einen Prompt erzeugen und die Routine mit einem einfachen Zusatzmodul auf einen folgenden

Tastendruck warten lassen.

Auch die ROM-Routine (bei&0556), die Daten vom Band lädt, wird zweimal aufgerufen: einmal für den Kopfsatz und ein zweites Mal für den Datenblock. Die folgende Tabelle zeigt die Bedingungen für den Einsprung und die Codeverifizierung:

Register	Laden	Verifizieren
C-Flag	1	0
A	0 für Kopfsatz, 255 für Datenblock	
IX	Zeigt auf die Speicheradresse, von der an die Bytes geladen werden sollen	
DE	Anzahl der zu ladenden Bytes; D muß im Bereich von 0 bis 254 liegen	

Die Routine belegt im Arbeitsspeicher 34 Bytes – den Platz für zwei Kopfsätze. Um eine Datei an eine bestimmte RAM-Adresse laden zu können, muß zunächst ein zweiter Kopfsatz mit den Steuerdaten der Banddatei angelegt werden. Dieser zweite Kopfsatz wird dann mit dem vom Band gelesenen Kopfsatz verglichen und so festgestellt, ob dies überhaupt die gewünschten Daten sind.

C-Flag auf Wert setzen

Beim Aufruf der Routine (bei &0556) muß das C-Flag auf den in der Tabelle aufgeführten Wert gesetzt werden. Codeabschnitte, die man verifizieren möchte, sollten bei der Adresse anfangen, auf die das IX-Register zeigt. Beim Verlassen der ROM-Laderoutine enthält das C-Flag das Ergebnis des Ladevorgangs. Steht C auf Eins, war das Laden erfolgreich. Wenn Sie jedoch einen Kopfsatz laden wollen und auf dem Band zuerst einen Datenblock finden, wird das C-Flag auf Null gesetzt. (Ladefehler werden vom OS abgefangen.) Hier ein Beispiel für das Laden eines Kopfsatzes:

```
        ;load a header from tape into RAM
        ;
37              scf             ;set carry I=LOAD
3E00            ld   a,0        ;0 indicates a header
DDE5            push ix         ;save ix on stack
DD2148EE        ld   ix,61000   ;load header to 61000
111100          ld   de,17      ;no of bytes in header
CD5605          call #0556      ;do it
DDE1            pop  ix         ;restore ix
C9              ret
```

Unmittelbar nach dem Laden wird der Kopfsatz mit dem im RAM angelegten Kopfsatz verglichen und überprüft, ob Dateiname, Dateityp und – falls nötig – die Länge des Blocks übereinstimmen. Das folgende Programm prüft nur den Namen. Falls er mit dem angegebenen identisch ist, wird der darauffolgende Datenblock bei der zuvor definierten Adresse ins RAM geladen.

```
        ;locate and load TESTPROG
        ;
DDE5            push ix         ;save IX on stack
37      loop:   scf             ;set carry for LOAD
3E00            ld   a,0        ;0=BASIC file
111100          ld   de,17      ;no of bytes in header
DD21CA2B        ld   ix,head2   ;load header into head2
CD5605          call #0556
D27B2B          jp   nc,loop    ;if no header then again
060A            ld   b,10       ;no of bytes in fname
11BA2B          ld   de,head+1  ;point to desired f/name
21CB2B          ld   hl,head2+1 ;point HL to found fname
1A      name:   ld   a,(de)     ;get header char in a
BE              cp   (hl)       ;comp with found header
2007            jr   nz,no      ;different so exit loop
13              inc  de         ;get next char
23              inc  hl
05              dec  b          ;decrement counter
20F7            jr   nz,name    ;same so check next char
1802            jr   ok         ;all chars checked + ok
18DB    no:     jr   loop       ;check failed, try again
        ;found right header so prepare to load data
DD21B92B ok:    ld   ix,head    ;get head in ix
DD6E0D          ld   l,(ix+13)  ;get address to
DD660E          ld   h,(ix+14)  ;load data into
E5              push hl         ;push address onto stack
DDE1            pop  ix         ;and get it into ix
3EFF            ld   a,255      ;indicate load data
        ;next instruction requires user to insert data
         length
11AF2B          ld   de,nn      ;insert data length
37              scf             ;indicate a LOAD
CD5605          call #0556      ;doit
DDE1            pop  ix         ;restore ix
C9              ret
        ;now follow details of desired f/name
03      head:   defb 3          ;indicates CODE block
54455354        defm "TESTPROG" ;f/name
00              defb 0          ;use for data length
00              defb 0          ;data-length hi-byte
48              defb 72         ;lo-byte for load
EE              defb 238        ;address of 61000
00              defb 0          ;used for BAS. prog only
00              defb 0          ; ditto
        ;now follows space for header loaded for
         checking
        head2:  defs 17
```

Auch hier werden keine Meldungen an den Schirm ausgegeben. Die Routine lädt einen Datenblock namens TESTPROG in den in HEAD angegebenen Speicherbereich. Zusätzlich sollte man prüfen, ob auch der Dateityp stimmt. Ähnlich wie bei LOAD und SAVE (in BASIC) können auch diese ROM-Routinen durch Drücken der Break-Taste einfach beendet werden.

Wir haben am Schluß dieser Folge ein kurzes Programm abgedruckt, das einen Kopfsatz vom Band liest und Informationen über die Datei auf dem Bildschirm ausgibt – Dateilänge, Anfangsadresse etc. Der Maschinencode ist in DATA-Befehlen untergebracht. Er lädt einen Kopfsatz und untersucht ihn ohne weitere Zwischenschritte von BASIC aus.

```
5 CLEAR 59999
10 FOR I=0 TO 19
20 READ A:POKE (60000+I),A
30 NEXT I
40 RANDOMIZE USR 60000
50 LET type=PEEK 60020
60 LET length=PEEK 60031+256*PEEK 60032
70 LET start=PEEK 60033+256*PEEK 60034
80 LET L$=""
90 FOR I=60021 TO 60030: LET L$=L$+CHR$(PEEK I): NEXT I
100 PRINT "Name: ";L$
110 IF type=0 THEN LET f$="BASIC"
120 IF type=1 THEN LET f$="Number Array"
130 IF type=2 THEN LET f$="String Array"
140 IF type=3 THEN LET f$="CODE"
150 PRINT "Type: ";f$
160 IF type=0 THEN PRINT "Auto Run line number: ";start
170 IF type<>0 THEN PRINT "Start address: ";start
180 PRINT "Length: ";length
190 GOTO 40
200 DATA 221,229,62,0,55,221,33,116,234,17,17,0,205,86,
    5,48,241,221,225,201
```

Hinter der Maske

Die Z80-CPU bietet drei maskierbare Interrupts, mit denen sich Maschinencodeprogrammierer Prozessorzeit „borgen" können. Wir untersuchen, welche Rolle Interrupts im Betriebssystem des Spectrum spielen und wie mit Vektoren eigene Maschinencodemodule angesprochen werden.

Da die interruptgesteuerten Module des Spectrum OS sehr komplex sind, sollten Sie vorsichtig damit umgehen. Ein unbeabsichtigt „abgeschaltetes" Interrupt kann das Funktionieren der Maschine behindern – es könnten beispielsweise keine Tastatureingaben mehr möglich sein – oder das System während der Ausführung eines BASIC-Programms abstürzen lassen.

Die Z80-CPU des Spectrum arbeitet mit zwei Arten von Interrupts – maskierbare und nicht maskierbare (NMI). Der Unterschied ist leicht beschrieben: Sie können die CPU so programmieren, daß sie maskierbare Interrupts ignoriert. Auf nicht-maskierbare Interrupts wird der Prozessor jedoch in jedem Fall reagieren.

Die NMI-Interrupts lassen sich auf dem Spectrum nicht besonders gut einsetzen, da die entsprechende Bearbeitungsroutine im ROM nicht zuverlässig arbeitet. Die Designer des OS wollten es dem Anwender ursprünglich ermöglichen, über die Speicherstellen &5CB0 und &5CB1 eine Adresse angeben zu können, die von der CPU bei Empfang eines NMI automatisch angesprochen wird. Der Z80 führt bei einem NMI jedoch standardmäßig den Maschinencode bei &66 aus und verursacht damit normalerweise einen Systemreset. Aus diesem Grund konzentrieren wir uns auf den Einsatz der maskierbaren Interrupts.

Die Z80-CPU besitzt mehrere Interruptmechanismen. Wir werden hier jedoch nur auf die Arten bzw. Modi eingehen, die für das OS des Spectrum Bedeutung haben.

50 Interrupts/sec

Bei der Initialisierung (d. h. beim Anschalten der Maschine oder bei Ausgabe des Befehls NEW) setzt die CPU automatisch den Interruptmodus 1 (IM1). Die Sinclair ULA (Uncommitted Logic Array – nicht festgelegte Anordnung von Logikschaltungen) liefert 50 Interruptimpulse pro Sekunde an die CPU. Im IM1-Modus führt die CPU bei jedem Empfang eines Interruptsignals den Befehl RST &0038 aus. Damit wird ein Sprung auf die Routine für Tastaturabfragen ausgelöst und der FRAMES-Zähler bei 23672 und 23674 inkrementiert. (Diese drei Bytes bilden einen 24-Bit-Zähler, der alle zwanzig Millisek. aktualisiert wird.)

Nach Interpretation einer BASIC-Zeile wartet der BASIC-Interpreter auf ein Interrupt, bevor er sich die nächste Zeile vornimmt. Wenn man die Interrupts abschaltet, bricht die BASIC-Programmausführung ab.

Nach Ausführung des Befehls DI (Disable Interrupts – Interrupts abschalten) ignoriert die CPU alle maskierbaren Interrupts. Der Befehl EI (Enable Interrupts) aktiviert sie wieder.

Im OS des Spectrum gibt es mehrere Routinen, bei deren Ausführung die Interrupts abgeschaltet sein müssen. Dies sind normalerweise zeitabhängige Routinen wie das BEEP-Modul des Tongenerators und die Lade- und Speicherroutinen der Cassettenstation.

Interrupts können auch von dem ZX Drucker, dem Interface 1 oder den Microdrives zeitweilig abgeschaltet werden. Nach Beendigung der Routine werden sie wieder aktiviert.

Während der Interruptabschaltung wird der FRAMES-Zähler nicht inkrementiert, das heißt, er „verliert Zeit". Da praktisch bei jedem Programmablauf Interrupts eintreten können, soll-

Für alle Fälle

Obwohl man fast immer davon ausgehen kann, daß der Datenbus beim Auftreten eines Interrupts ein 255 enthält, gibt es Peripheriegeräte, die diesen Wert ändern. Das Problem tritt nicht auf, wenn Sie den Interruptmodus 2 (IM2) ohne Peripheriegeräte verwenden.

Als alternative Methode, alle nur möglichen Werte des Datenbusses abzufangen, füllen sie eine Speicherseite mit identischen Werten und laden vor dem Anschalten von IM2 die Seitennummer in das I-Register. Beim Eintreten eines Interrupts holt sich der Z80 die Adresse der Bearbeitungsroutine aus einer beliebigen Position der angegebenen Seite – die exakte Position bestimmt der Datenbus.

Zu diesem Zweck wird das I-Register zunächst mit &FC geladen und die Bytes von &FC00 bis &FD00 auf den Wert &FB gesetzt. Danach schalten Sie IM2 an. Wenn der Datenbus beim nächsten Interrupt &C3 enthält, holt der Z80 sich die Adresse der Interrupt-Bearbeitungsroutine aus der Speicherstelle &FCC3 – den &FC-Teil der Adresse liefert das I-Register. Da alle 256 Bytes der Seite &FC (und Byte Null der Seite &FD) auf dem gleichen Wert stehen, lautet die Vektoradresse in jedem Fall &FBFB.

Dazu noch zwei wichtige Anmerkungen: Die Interrupt-Bearbeitungsroutine muß immer bei einer Adresse mit identischen Lo- und Hi-Bytes liegen (z. B. &C4C4 oder &FDFD). Berücksichtigen Sie weiterhin die Möglichkeit, daß der Adreßbus beim Auftreten eines Interrupts &FF enthält. In diesem Fall sucht der Z80 die Bearbeitungsroutine bei den Adressen nnFF (Lo-Byte) und (nn+1)00 (Hi-Byte) – (nn ist der Wert des I-Registers). Setzen Sie daher immer auch das erste Byte der darauffolgenden Seite.

ten Sie die Interrupts bei zeitkritischen Modulen möglichst abschalten. Vor der Rückkehr ins BASIC müssen sie jedoch unbedingt wieder aktiviert werden.

Um Interrupts praktisch einsetzen zu können, muß zuerst ein flexibler Interruptmodus eingestellt werden. Dafür eignet sich am besten der Interruptmodus IM2, der flexibler als IM1 ist. Während IM1 immer mit RST auf die Adresse &0038 springt, läßt sich IM2 auf jeden beliebigen Interruptvektor leiten.

Der Interruptvektor enthält die Startadresse der Bearbeitungsroutine, die beim Auftreten eines maskierbaren Interrupts ausgeführt wird. Dabei zeigt das I-Register der CPU an, wo sich der Vektor überhaupt befindet. Die Adresse der Interrupt-Vektorroutine ergibt sich aus dem Inhalt des I-Registers und des Datenbusses zum Interruptzeitpunkt. In einigen Systmen setzt die Interruptquelle außerdem ein Byte auf den Datenbus, das der CPU mitteilt, welches Gerät die Unterbrechung ausgelöst hat.

Die Sinclair ULA arbeitet nicht mit dieser Methode, doch ist der Spectrum so konstruiert, daß der Datenbus den Wert 255 enthält, wenn sich keine weitere Eingabe darauf befindet. Das I-Register liefert nun das höherwertige Byte der 16-Bit-Adresse des Interruptvektors und der Datenbus das niederwertige (in diesem Fall &FF). So befindet sich der Interruptvektor immer an der Grenze einer Speicherseite: Das niederwertige Adreßbyte liegt in &nnFF und das höherwertige in &(nn+1)00 (nn ist der Inhalt des I-Registers).

Wenn I beispielsweise den Wert &FB enthält, befindet sich die Vektoradresse bei &FBFF. Das niederwertige Vektorbyte (bei &FBFF) liefert dann das Lo-Byte der Interrupt-Bearbeitungsroutine und &FC00 das Hi-Byte.

Achtung! Hardwareprobleme

Bearbeitungsroutinen für Interrupts dürfen nicht an jeder beliebigen Speicherposition liegen – so sind zum Beispiel die ersten 16 KBytes dem ROM zugeordnet und stehen damit nicht zur Verfügung, um hier Routinen unterzubringen, die die Interruptsteuerung beeinflussen. Außerdem können Hardwareprobleme auftreten, wenn der Wert des I-Registers zwischen 64 und 127 liegt.

Sehen wir uns nun an, wie der Interruptmodus 2 angeschaltet und der Interruptvektor mit der Adresse der Bearbeitungsroutine eingesetzt wird:

```
F3          di                  ;disable interrupts
210000      ld   hl,ADDRESS     ;get ISR address in HL
22FFFB      ld   (#FBFF),hl     ;get address into vector
3EFB        ld   a,#FB          ;hi-byte of vector...
ED47        ld   i,a            ;...into I register
ED5E        IM   2              ;set int mode 2
FB          ei                  ;re-enable interrupts
C9          ret                 ;back to BASIC
```

Das Listing setzt natürlich voraus, daß sich bei ADDRESS eine Routine befindet, die den Interrupt bearbeiten kann – ist dies nicht der Fall, stürzt das Programm ab. Idealerweise sollte die Bearbeitungsroutine auch alle Aufgaben übernehmen, die sonst vom normalen Interruptmodus des Spectrum ausgeführt werden. Dies geschieht am besten über einen Aufruf der Routine bei &38. Das folgende Assemblerprogramm stellt den Interruptmodus auf IM2 um und veranlaßt die CPU, bei jedem Interrupt die Routine bei ADDRESS auszuführen (die hier nur die üblichen Interruptfunktionen des Spectrum erledigt).

```
            org  60000             ;specify start add
     vector: equ  #FEFF            ;FEFF is vector add
216FEA change: ld   hl,addres      ;get address into HL
22FFFE      ld   (vector),hl       ;set up vector
F3          di                     ;disable interrupts
3EFE        ld   a,#FE             ;set up the...
ED47        ld   i,a               ;...I register
ED5E        im   2                 ;change int mode
FB          ei                     ;re-enable interrupts
C9          ret                    ;back to BASIC
F3     addres: di                  ;turn off interrupts
FF          rst  #38               ;normal IM1 procedure
FB          ei                     ;re-enable interrupts
C9          ret
```

Der Aufruf der Routine bei Adresse CHANGE setzt den neuen Interruptmodus und den Vektor. Danach wird die Routine bei ADDRESS alle Fünfzigstelsekunde angesprochen. Im Augenblick erledigt diese Routine zwar nur die üblichen Standardaufgaben, doch werden wir in Kürze einige interessante Funktionen dafür programmieren.

Die folgende Routine kann jederzeit den normalen Interruptmodus wiederherstellen:

```
F3          di
3E3F        ld   a,#3f             ;reset I register... '
ED47        ld   i,a               ;...to its usual value
ED56        im   1                 ;normal mode
FB          ei                     ;re-enable interrupts
C9          ret
```

Unseren eigenen Module werden als Subroutinen angelegt und mit CALL aufgerufen:

```
F3     addres: di
F5             push af             ;save registers...
C5             push bc
D5             push de
E5             push hl
FF             rst  #38            ;call normal ISR
F3             di                  ;ISR did EI, so DI again
CD0000         call PROG_ADD       ;call our own routine
E1             pop  hl             ;restore registers...
D1             pop  de
C1             pop  bc
F1             pop  af
FB             ei
C9             ret
```

Interruptroutinen verlangsamen den Spectrum. Denken Sie auch daran, die Interrupts am Anfang Ihrer eigenen Interruptroutinen auszuschalten, damit während der Ausführung keine weitere Unterbrechung eintreten kann.

Musik mit Unterbrechung

Der Tondatencompiler stellt ein Menü mit drei Bearbeitungsmöglichkeiten dar. „C-COMPILE“ verwandelt die Melodie (in den DATA-Befehlen zwischen Zeile 10 und 900 gespeichert) in einen Codeblock, der von der Interruptroutine eingesetzt wird. ‚P‘ spielt Ihnen die Melodie zur Prüfung vor. ‚R‘ verzweigt ins BASIC zurück.

Beachten Sie, daß Sie bei Option ‚C‘ angeben müssen, in welchem Speicherbereich Sie die Tondaten unterbringen wollen. Speichern Sie sie oberhalb von RAMTOP (der Wert muß bereits geändert sein, damit Platz für die Daten frei ist). Wenn die Töne compiliert und gespeichert sind, werden die Basisadresse und die Programmlänge angezeigt. Am besten notieren Sie sich diese beiden Werte, da Sie sie später noch brauchen. Zur Speicherung des Codes geben Sie von BASIC aus SAVE „NOTEDATA“ CODE (Basisadresse), (Programmlänge in Bytes) ein. Für den Einsatz Ihrer eigenen Melodien brauchen Sie nun nur noch die Zeilen 10 bis 90 zu löschen und eigene Daten zwischen den Zeilen 10 und 900 zu speichern.

Basisadresse der Routine ist 65021. Stellen Sie daher vor dem Laden des Codes sicher, daß RAMTOP heruntergesetzt wurde (CLEAR 65000). Nach Assemblierung und Laden des Programmcodes übertragen Sie den vom Tondatencompiler erzeugten Code in den Speicher (siehe unten). Stellen Sie auch hier sicher, daß über RAMTOP ausreichend Platz zur Verfügung steht.

10 LET L=65152: LET V=(* Basisadresse der Tondaten *)
20 POKE L+1,INT (V/256):
POKE L,V-
PEEK(L+1)*256

RAND USR 65041 schaltet nun auf IM2, und die Musik fängt an zu spielen. RAND USR 65071 aktiviert wieder IM1 und stoppt die Musik.

Tondatencompiler

```
  1 REM >>> INTERRUPT MUSIC <<<
  2 REM >>Note Data  Compiler<<
  3 REM
  6 REM *NOTE DATA IN STANDARD SPECTRUM BEEP FORM
AT *
  7 REM
  9 RESTORE
 10 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,
1,9,2,10,1,10,2,10,1,12,5,14
 20 DATA 1,14,2,7,1,7,1,7,1,6,1,7,2,16,1,14,2,14,
1,10,2,9,1,9,2,9,1,10,5,12
 30 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,
1,12,2,11,1,19,2,19,1,19,5,19
 40 DATA 1,17,2,16,1,19,1,19,1,18,1,19,2,14,1,19,
1,19,1,18,1,19,2,12,1,11,2,12,1,11,3,12
 50 DATA 3,10,1,9,1,8,1,9,2,14,1,12,3.2,9,3.2,5,3
.2,2,3.2,7,5,5
 60 DATA 1,5,1,7,1,9,1,10,2,16,1,14,3.2,12,3.2,17
,3.2,16,3.2,14,5,12
 70 DATA 1,12,2,14,1,14,1,14,1,13,1,14,3,16,3,9
 80 DATA 2,17,1,17,1,19,1,17,1,19,5,21
 90 DATA 1,21,2,19,1,17,2,14,1,10,3.2,9,3.2,5,3.2
,7,3.2,4,5,5
 997 REM
 998 REM *DO NOT ERASE LINE 999*
 999 DATA 255,255
1000 REM
2000 REM **** MENU ****
2010 BORDER 1: PAPER 1: INK 6
2020 CLS
2030 PRINT AT 5,4;"INTERRUPT MUSIC COMPILER";AT 10
,5;"C -- Compile note data";AT 12,5;"P -- Play tun
e in beeps";AT 14,5;"R -- Return to BASIC"
2040 LET a$=INKEY$
2050 IF a$="c" OR a$="C" THEN  GO TO 5000
2060 IF a$="p" OR a$="P" THEN  GO TO 3000
2070 IF a$="r" OR a$="R" THEN  CLS : STOP
2080 GO TO 2040
3000 RESTORE
3010 READ d,f: IF d=255 AND f=255 THEN  GO TO 2000
3020 BEEP d/10,f-6: GO TO 3010
5000 RESTORE : CLS : INPUT "BASE ADDRESS OF NOTE D
ATA? ";dd1: POKE 23301,INT (dd1/256): POKE 23300,d
d1-(256*PEEK 23301): CLEAR dd1-1: LET dd1=PEEK 233
00+256*PEEK 23301
5005 LET f=0: LET d=dd1
5010 READ delay,pitch
5015 LET delay=INT (delay*6)
5020 LET freq=(1.0594631^pitch)*256
5030 LET bip=INT ((437500/freq)-30.125)
5035 IF delay=255*6 THEN  POKE d,255: GO TO 6000
5037 LET f=f+1: PRINT AT 10,10;"NOTE )";f
5040 POKE d,delay
5060 POKE d+1,bip-(INT (bip/256)*256)
5070 POKE d+2,INT (bip/256)
5080 LET d=d+3
5090 GO TO 5010
6000 PAUSE 50: CLS : LET len=(d+2)-dd1
6010 PRINT "LOCATION OF NOTE DATA IN MEMORY=";dd1'
''"NUMBER OF BYTES OF NOTE DATA....";len
6020 INPUT "PRESS 'ENTER' TO RETURN TO MENU "; LIN
E a$: GO TO 2000
```

Ladeprogramm in BASIC

```
 10 RESTORE 9000
 20 CLEAR 64000
 30 LET cqs=0: FOR f=65021 TO 65156
 40 READ dat: POKE f,dat
 50 LET cqs=cqs+dat
 60 NEXT f
 70 IF cqs<>18850 THEN  PRINT "ERROR IN DATA !!!"
: STOP
 80 PRINT "OK NO PROBLEMS IN DATA"
 90 STOP
9000 DATA 0,0,221,229,229,197,213,245,195,52,254,2
41,209,193,225,221,225,195,56,0,33,255,253,34,253,
253,243,62,253,237,71,237,94,42,128,254,34,126,254
,62,1,50,130,254,61,50,131,254,251,201,243,237,86,
251,201,58,130,254,254,0,202,8,254,58,131,254,254,
0,202,75,254,61,50,131,254,195,8,254,42,126,254,12
6,254,255,202,108,254,50,131,254,35,126,35,95,126,
35,34,126,254,103,107,17,4,0,205,181,3,243,195,8,2
54,42,128,254,34,126,254,62,1,50,130,254,61,50,131
,254,195,8,254,0,0,0,0,0,0,
```

Assemblerlisting

```
                org  65021
0000    vect:   defw 0                  ;2 bytes for vector
DDE5    inter:  push ix
E5              push hl
C5              push bc
D5              push de
F5              push af
C334FE          jp   start              ;jp to music routine
F1      enprg:  pop  af
D1              pop  de
C1              pop  bc
E1              pop  hl
DDE1            pop  ix
C33800          jp   56                 ;jp to ROM ISR
21FFFF  on:     ld   hl,inter           ;set up vector
22FDFD          ld   (vect),hl
F3              di
3EFD            ld   a,253
ED47            ld   i,a                ;get vect hi-byte in I
ED5E            im   2                  ;set int mode 2
2A00FE  initi:  ld   hl,(datad)         ;point to data
227EFE          ld   (locat),hl
3E01            ld   a,1                ;switch = 1 = play note
3282FE          ld   (swtch),a
3D              dec  a                  ;delay = 0
3283FE          ld   (delay),a
FB              ei
C9              ret                     ;back to BASIC
F3      off:    di
ED56            im   1                  ;set int mode 1
FB              ei
                ret
C9      ;
3A82FE  start:  ld   a,(swtch)          ;get switch in a
FE00            cp   0
CA08FE          jp   z,enprg            ;if swtch 0 then end
3A83FE          ld   a,(delay)
FE00            cp   0
CA4BFE          jp   z,nwnot            ;if delay 0 then play
3D              dec  a                  ;delay=delay-1
3283FE          ld   (delay),a          ;store new delay value
C308FE          jp   enprg              ;goto end routine
2A7EFE  nwnot:  ld   hl,(locat)         ;get data location
7E              ld   a,(hl)             ;get next delay in a
FEFF            cp   255                ;255 = data finished
CA6CFE          jp   z,reset            ;data ended so reset
3283FE          ld   (delay),a          ;store new delay
23              inc  hl                 ;point to next databyte
7E              ld   a,(hl)             ;get it in a
23              inc  hl                 ;point to next databyte
5F              ld   e,a                ;store byte1 in e
7E              ld   a,(hl)             ;store byte2 in a
23              inc  hl                 ;point to next byte
227EFE          ld   (locat),hl         ;store next-byte-address
67              ld   h,a                ;get frequency data...
68              ld   l,e                ;...in HL
110400          ld   de,4               ;note duration of 4
CDB503          call 949                ;call ROM beep routine
F3              di                      ;ROM-BEEP did EI, so DI
C308FE          jp   enprg              ;goto end routine
2A00FE  reset:  ld   hl,(datad)         ;reset various...
227EFE          ld   (locat),hl         ;...pointers
3E01            ld   a,1
3282FE          ld   (swtch),a
3D              dec  a
3283FE          ld   (delay),a
C308FE          jp   enprg              ;goto end routine
0000    locat:  defw 0                  ;reserve memory for...
0000    datad:  defw 0                  ;various pointers...
00      swtch:  defb 0
00      delay:  defb 0
```

Ein ROM im Schatten

Das Interface 1 gibt dem Sinclair Spectrum nicht nur Zugang zu den Microdrives, dem lokalen Netzwerk (LAN) und der seriellen Schnittstelle, sondern ermöglicht auch Maschinencoderoutinen. Wir sehen uns einige dieser Routinen genauer an.

Das Interface besitzt ein ROM mit acht KBytes, die den Anfangsbereich des Speichers belegen. Der als „Schatten-ROM" bezeichnete Chip enthält Routinen zur Steuerung der Zusatzgeräte (z. B. der Microdrives) und einen BASIC-Interpreter, der Befehle wie CAT und FORMAT entgegennimmt.

Es existieren mehrere Schatten-ROM Versionen, die zwischen Version 1 und den neueren Ausgaben (Version 2) wesentliche Unterschiede aufweisen. Version 1 konnte beispielsweise die Microdrives nicht optimal einsetzen.

Spätere Versionen sprechen die Microdrives effektiver an. Weiterhin wurden Fehler bereinigt und neue Fähigkeiten für serielle Drucker eingebaut.

Für den Anwender sind die Unterschiede zwischen den Versionen normalerweise „transparent" – vorausgesetzt, das Interface wird über BASIC oder mit den korrekten Maschinencodetechniken angesprochen. Wenn Sie die Routinen des Schatten-ROM direkt aufrufen, werden Sie mit Sicherheit Schwierigkeiten bekommen, da die Routinen von Version zu Version unterschiedliche Adressen benutzen. Alle Adreßangaben dieser Serie beziehen sich auf die Version 1 des Schatten-ROMs.

ROM ein, ROM aus

Wie kann nun das Schatten-ROM die ersten acht Speicherbytes belegen, wenn sich dort ebenfalls das BASIC-ROM befindet? Hier wird mit einer Technik gearbeitet, die dem System der Paged ROMs des Acorn B entspricht. Sobald der Z80 versucht, Befehle aus den Adressen &08 oder &1708 zu lesen, wird das BASIC-ROM „abgeschaltet" und das Schatten-ROM aktiviert. Das ROM kann mit einer eigenen Routine das BASIC-ROM wieder einschalten.

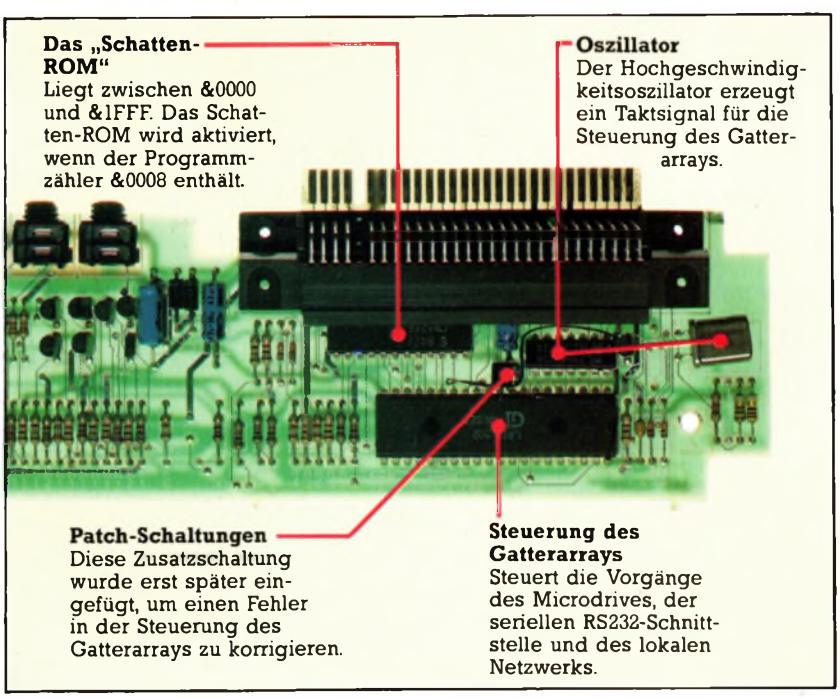
Das Interface 1 muß vor dem Einsatz jedoch erst initialisiert werden. Eine Initialisierung ist nötig, wenn

1) der Befehl NEW ausgegeben oder
2) das Schatten-ROM zum ersten Mal aktiviert wird.

Die Initialisierung legt im RAM eine neue Variablengruppe (etwa 50 Bytes) an, die vom Ende der alten Systemvariablen bis zum Anfang der Kanaldaten reicht. Da außerdem eine „Microdrive-Map" gebraucht wird und weitere Informationen über die von dem Interface 1 gesteuerten Peripheriegeräte nötig sind, verschiebt sich der Start der BASIC-Programmtexte nach oben. Aus diesem Grund sollten Sie Maschinencode auch nicht in REM-Befehlen der Zeile 1 unterbringen, da diese Zeile (bei eingebautem Interface 1) nicht mehr auf einer festen Speicherposition liegt.

Das Interface 1 enthält zusätzliche BASIC-Befehle und gibt Ihnen auch die Möglichkeit, eigene Befehle zu generieren. Auf einem Spectrum (ohne Interface 1) rufen die Befehle CAT und FORMAT Fehlermeldungen hervor, die über den Befehl RST &08 die Adresse &08 ansprechen. Mit dem Interface 1 aktiviert dieser Befehl jedoch das Schatten-ROM, das sich den „beanstandeten" BASIC-Befehl nochmals ansieht und versucht, ihn zu interpretieren. Der folgende Befehl speichert auf diese Weise das Programm „Fred" auf dem Microdrive:

```
SAVE *„m";1;„Fred"
```



Das Bild zeigt die Platine des Interface 1. Außer den Routinen zur Steuerung der Microdrives, der seriellen Schnittstelle und des LAN können Sie damit auch eigene BASIC-Befehle definieren. Wir gehen in einer späteren Folge genauer auf diesen Vorgang ein.

Der * erzeugt eine Fehlerbedienung, die die Steuerung an die Adresse &08 und damit an das Schatten-ROM übergibt. Dort erfolgt dann die korrekte Interpretation des Befehls.

Sehen wir uns nun die Maschinencoderoutinen an, die das Interface 1 zu bieten hat. Wir untersuchen zunächst die „Allzweckroutinen" – das heißt die Module des Schatten-ROMs, die nicht zur Steuerung der Microdrives, der seriellen Schnittstelle und des LAN eingesetzt und benötigt werden.

Routinen – aber wie?

Zunächst stellt sich die Frage, wie die Routinen überhaupt aufgerufen werden, da das Schatten-ROM doch nur nach Ansprechen einder der beiden zuvor erwähnten Adressen aktiv ist. Die Lösung ist ein Sprung auf die Adresse &08 mit dem Ablauf

```
RST &08
DEFB nn
```

wobei nn einen Wert zwischen &1B und &32 angibt (Werte außerhalb dieses Bereichs erzeugen eine Fehlermeldung). Die Werte von nn heißen „Hakencodes" (englisch: Hook Codes) und rufen unterschiedliche Routinen des Schatten-ROM auf. Die Auswirkungen der Hakencodes wurden in allen Versionen des Schatten-ROM unverändert beibehalten. Bevor wir uns einige dieser Aufrufe ansehen, hier noch ein paar Hinweise:

1) Da Vorgänge mit Hakencodes die Tendenz haben, Registerinhalte zu zerstören, müssen Sie vor dem Ablauf alle Register sichern, die Sie später noch einsetzen wollen. Auch das Registerpaar HL sollte gesichert werden, da es für die Rückkehr ins BASIC unbedingt nötig ist.

2) Einige Hakencodes schalten die Interrupts ab. Im Zweifelsfall sollten Sie sie wieder anschalten.

3) Beim Aufruf eines Hakencodes sollte der Wert &5C3A im Registerpaar IY stehen.

4) Vor dem Einsatz eines Hakencodes sollten Sie sicherstellen, daß die Systemvariablen des Interface aktiv sind. Diese Aufgabe kann der zuerst beschriebene Hakencode übernehmen.

- **Hakencode 49 (&31)** veranlaßt die Anlage der Systemvariablen für das Interface 1. Wenn Sie nicht genau wissen, ob die Variablen bereits angelegt sind, sollten Sie vor dem Einsatz anderer Hakencodes folgenden Aufruf durchführen:

```
RST 8
DEFB 49
```

Nach Ausführung eines Hakencodes durch das Schatten-ROM geht die Steuerung wieder an das Haupt-ROM zurück.

Das Schatten-ROM bietet aber auch Routinen, die (in komplizierterer Form) im BASIC ROM enthalten sind und die Tastatur-E/A und die Bildschirmausgabe steuern.

- **Hakencode 27 (&1B)** wartet auf die Eingabe eines Zeichens per Tastatur und übergibt den Tastencode an das Register A (das Zeichen erscheint nicht auf dem Bildschirm).
- **Hakencode 32 (&20)** prüft unmittelbar nach seinem Aufruf die Tastatur und zeigt über den Status des C-Flag an, ob eine Taste gedrückt wurde. Die Routine wartet jedoch nicht auf einen Tastendruck. Wurde eine Taste betätigt, dann steht das C-Flag auf Eins; wenn nicht, steht es auf Null.

Vor dem Aufruf der Hakencodes 27 und 32 müssen die Interrupts aktiviert werden, da der Spectrum das Prüfen der Tastatur über Interrupts steuert.

- **Hakencode 28 (&1C)** setzt das (beim Aufruf der Routine als ASCII-Code im A-Register gespeicherte) Zeichen in den Strom 2, der normalerweise dem oberen Teil des Bildschirms zugeordnet ist. Das folgende Programmodul zeigt den Einsatz dieser Routine und des Hakencodes 27. Beachten Sie, daß hier eine Endlosschleife vorliegt, die nur durch Abschalten der Maschine beendet werden kann.
- **Hakencode 31 (&1F)** ähnelt dem Hakencode 28, setzt das Zeichen (den ASCII-Code im A-Register) aber in Strom 3 (normalerweise dem ZX Drucker zugeordnet).
- **Hakencode 50 (&32)** spricht eine Allzweckroutine an. Der Code aktiviert vom Haupt-ROM aus Routinen des Schatten-ROMs über ihre direkten Adressen und nicht über den Hakencode. Die Routine verursacht jedoch einen Systemabbruch, wenn sie auf einer anderen ROM-Version eingesetzt wird. Hier ihr Einsatz:

```
      ;call routine in Shadow Rom - use with care!
210000      ld   hl,ADDRESS          ;insert address of routine
22ED5C      ld   (23789),hl          ;put address in system variable
CF          rst  #08
32          defb #32                 ;do it
```

Zwei Systemadressen des Interface 1 (23789 und 23790) müssen die Adresse enthalten, die die gewünschte Routine des Schatten-ROMs angibt. Danach wird der Hakencode folgendermaßen aufgerufen:

```
      ;wait for char, print to stream 2
      ;Hook Codes #1B and #1C
CF    start: rst  #08                ;set up Interface 1...
31           defb #31                ;...system variables
FB    loop:  ei                      ;ei - just in case
CF           rst  #08                ;wait for a char...
1B           defb #1B                ;...from the keyboard
CF           rst  #08                ;char now in A reg...
1C           defb #1C                ;...so print it
18F9         jr   loop               ;round again ad infinitum
```

Dieser Aufruf ist nur dann sinnvoll, wenn Sie wissen, mit welcher Version Sie arbeiten und welche Aufgaben die Routinen des Schatten-ROMs ausführen. Wir werden genauer darauf eingehen, wenn wir mit dem Interface 1 das BASIC um neue Befehle erweitern. In der nächsten Folge beschäftigen wir uns mit den Hakencodes, die die Microdrives steuern.

Gesteuerte Treiber

In der letzten Folge haben wir uns mit den „Hakencodes" des Interface 1 beschäftigt. Diesmal sehen wir uns die Routinen an, mit denen die Microdrives des Spectrum gesteuert werden. Viele dieser Codes lassen sich in der Maschinenprogrammierung einsetzen.

Die Microdrives werden entweder indirekt über Hakencodes oder direkt mit den ROM-Routinen des Interface 1 angesprochen. Wir gehen hier jedoch nur auf die Codes ein, die in Verbindung mit dem Microdrive eingesetzt werden.

Zuvor ein kurzer Blick auf die Funktionsweise der Microdrives. Eine formatierte Cartridge enthält 255 Sektoren, in denen die Datenblöcke physisch gespeichert werden. Jeder Sektor besteht aus einem Kopf (mit dem Cartridgenamen und anderen Informationen), einem Datenblock (für den Namen der Datei, die später hier gespeichert wird), weiteren Daten und 512 Datenbytes. Der Formatiervorgang markiert außerdem alle Cartridgesektoren, die nicht beschreibbar sind.

Die Datenübertragung zwischen Microdrives und Spectrum läuft über einen Kanal, dem 595 Bytes zur Verfügung stehen. Ihm ist eine „Microdrive-Map" mit 32 Bytes zugeordnet, die den Status der Sektoren anzeigt. Die Microdrive-Map gibt an, welche Sektoren bereits belegt oder nicht einsetzbar sind.

Beim Anlegen eines neuen Kanals wird der Speicher zwischen dem Ende des Kanalbereichs für Microdrives und STKEND nach oben verschoben, um Platz zu schaffen. Nach dem Schließen eines Kanals wird der Speicher wieder entsprechend nach unten verlegt.

Microdrivebefehle

Das System hat zwei Möglichkeiten, Microdrivekanäle anzulegen: direkt – beim Eröffnen einer Datei mit dem Befehl OPEN und indirekt – bei einem Systemablauf, der einen Kanal automatisch öffnet. So eröffnet der Microdrive-Befehl SAVE auf indirekte Weise einen Kanal, der später möglicherweise nicht direkt eingesetzt wird. Direkt eröffnete Kanäle müssen duch einen weiteren Befehl wieder geschlossen werden, indirekt eröffnete Kanäle erledigen dies automatisch.

Das nebenstehende Bild zeigt die Struktur eines Microdrivekanals. Die Anfangsbytes dieses Kanals stimmen mit den Anfangsbytes der Kanäle überein, die Daten an den Bildschirm senden, oder von der Tastatur empfangen. Die meisten dieser Bereiche werden jedoch nur selten direkt angesprochen.

Da im Kanal auch die Adresse der Microdrive-Map gespeichert ist, läßt sich leicht feststellen, wieviel Platz auf der Cartridge noch zur Verfügung steht. Auch der BASIC-Befehl CAT arbeitet mit der Microdrive-Map, die Sie sich mit folgender Routine anzeigen lassen können (geben Sie zuerst NEW ein):

```
 10 OPEN #5;"M";1;"testprog"
 20 start=PEEK(23870)+256*PEEK(23871)
 30 FOR I=start TO start+31
 40 LET sector=PEEK(I)
 50 FOR J=1 TO 8
 60 IF sector/2=INT(sector/2) THEN PRINT "0";
 70 IF sector/2<>INT(sector/2) THEN PRINT "1";
 80 LET sector=sector/2:LET sector=INT(sector)
 90 NEXT J
100 PRINT
110 NEXT I
120 CLOSE #5
```

Beim Programmablauf zeigt eine 1 einen nicht einsatzbereiten oder bereits beschriebenen Sektor an und eine 0 einen freien. Über eine kleine Programmänderung kann man sich auch den gesamten auf der Cartridge zur Verfügung stehenden Platz anzeigen lassen.

Das OS füllt die 512 Bytes des Datenbuffers mit Informationen, die auf Band geschrieben werden sollen. Bei einem gefüllten Buffer oder bei Ausgabe des Befehls CLOSE (oder des entsprechenden Hakencodes) läuft automatisch der Schreibvorgang ab. Im zweiten Fall erhält der Datensatz dabei die Markierung EOF (End Of File – Dateiende). EOF kann aber auch über einen Hakencode geschrieben werden – wir kommen später darauf zurück. Normalerweise wird die Cartridge in Abschnitten von 512 Bytes beschrieben.

Sehen wir uns nun die Hakencodes genauer an, die die Microdrives steuern.

- Hakencode 33 – startet den Motor des angegebenen Microdrives – die Drives haben die Nummern Eins bis Acht. Die Routine kann weiterhin die Motoren aller Microdrives gleichzeitig abstellen. Da nach dem Ablauf möglicherweise die Interrupts abgeschaltet sind, sollten Sie sie in jedem Fall mit dem Befehl „EI" reaktivieren. Für den Einsatz der Routine setzen Sie die Nummer des Drives, den Sie anschalten wollen, in das Register A und rufen mit RST#&8 den Hakencode auf. Der Wert 0 im Register A schaltet alle Drives ab. Dieser Ablauf läßt sich

jedoch auch mit einem Fehlercode des BASIC auslösen. Hakencodes können Fehler auf zwei Weisen erzeugen: Sie versuchen einen Drive anzuschalten, der nicht angeschlossen ist, oder sie sprechen eine Drive an, der keine formatierte Cartridge enthält.

● **Hakencode 34** – eröffnet eine Cartridgedatei. Die Routine prüft zunächst, ob auf der Cartridge eine Datei mit dem angegebenen Namen bereits existiert. Gibt es den Namen schon, dann wird die Datei zum Lesen eröffnet, gibt es ihn nicht, wird sie zum Schreiben vorbereitet. Das Schreib/Lese-Flag des angesprochenen Kanals zeigt mit Bit 0 auf 0 das Lesen und mit Bit 0 auf 1 das Schreiben einer Datei an.

Eine Datei kann mehrmals zum Lesen eröffnet werden. Dabei wird auch dann ein neuer Kanal belegt, wenn die zuvor eröffneten Kanäle noch aktiv sind.

Variablen vorhanden?

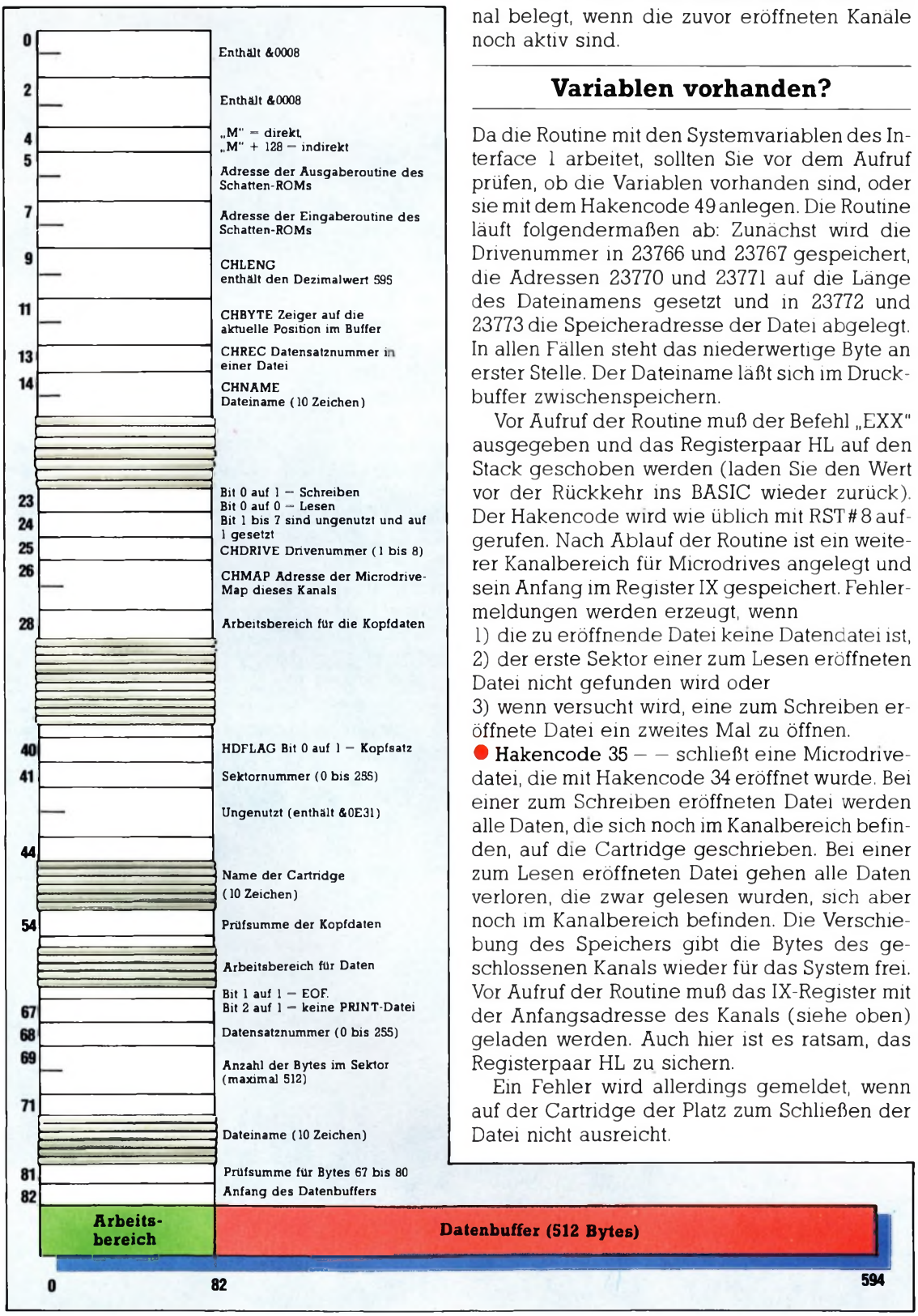
Da die Routine mit den Systemvariablen des Interface 1 arbeitet, sollten Sie vor dem Aufruf prüfen, ob die Variablen vorhanden sind, oder sie mit dem Hakencode 49 anlegen. Die Routine läuft folgendermaßen ab: Zunächst wird die Drivenummer in 23766 und 23767 gespeichert, die Adressen 23770 und 23771 auf die Länge des Dateinamens gesetzt und in 23772 und 23773 die Speicheradresse der Datei abgelegt. In allen Fällen steht das niederwertige Byte an erster Stelle. Der Dateiname läßt sich im Druckbuffer zwischenspeichern.

Vor Aufruf der Routine muß der Befehl „EXX" ausgegeben und das Registerpaar HL auf den Stack geschoben werden (laden Sie den Wert vor der Rückkehr ins BASIC wieder zurück). Der Hakencode wird wie üblich mit RST # 8 aufgerufen. Nach Ablauf der Routine ist ein weiterer Kanalbereich für Microdrives angelegt und sein Anfang im Register IX gespeichert. Fehlermeldungen werden erzeugt, wenn

1) die zu eröffnende Datei keine Datendatei ist,
2) der erste Sektor einer zum Lesen eröffneten Datei nicht gefunden wird oder
3) wenn versucht wird, eine zum Schreiben eröffnete Datei ein zweites Mal zu öffnen.

● **Hakencode 35** – – schließt eine Microdrivedatei, die mit Hakencode 34 eröffnet wurde. Bei einer zum Schreiben eröffneten Datei werden alle Daten, die sich noch im Kanalbereich befinden, auf die Cartridge geschrieben. Bei einer zum Lesen eröffneten Datei gehen alle Daten verloren, die zwar gelesen wurden, sich aber noch im Kanalbereich befinden. Die Verschiebung des Speichers gibt die Bytes des geschlossenen Kanals wieder für das System frei. Vor Aufruf der Routine muß das IX-Register mit der Anfangsadresse des Kanals (siehe oben) geladen werden. Auch hier ist es ratsam, das Registerpaar HL zu sichern.

Ein Fehler wird allerdings gemeldet, wenn auf der Cartridge der Platz zum Schließen der Datei nicht ausreicht.



Der Microdrivekanal belegt 595 Speicherbytes. Nach Eröffnung einer Cartridgedatei befindet sich die Anfangsadresse des Kanalbereichs im Register IX.

● **Hakencode 36** – – löscht eine Microdrivedatei – unabhängig davon, ob sie Daten oder Programme enthält. Dateiname, Adresse, Länge und Drivenummer werden wie beim Hakencode 34 angelegt.

Wie läßt sich eine eröffnete Datei nun beschreiben oder lesen? Microdrivekanäle lassen sich zwar jedem Strom zuordnen, doch ist es am einfachsten, den Microdrivekanal als „aktuellen" Kanal für das Lesen und Schreiben zu definieren.

Ist ein Microdrivekanal erst einmal als Ausgabekanal definiert, können Sie mit RST# 10 leicht Daten darauf schreiben. Dazu muß die Systemvariable CURCHL (bei 23633 und 23634) auf die Anfangsadresse des Microdrivekanals gesetzt werden. Hier der Befehl:

LD (23633), IX

(In IX ist die vom Hakencode 34 gelieferte Anfangsadresse des Kanals gespeichert.) RST# 10 schreibt nun seine Datenbytes in den Microdrivekanal statt sie zum Bildschirm zu senden. Hier zeigt sich, wie flexibel das Strom- und Kanalsystem des Spectrums ist – ein Aspekt, der nur selten berücksichtigt wird. Sobald der Buffer voll ist, wird er auf die Cartridge übertragen. Die folgende Routine schreibt die Datei FRED:

```
                ;write file "FRED" to microdrive
3E01            ld   a,1                ;drive number
32D65C          ld   (23766),a          ;store drive number
AF              xor  a                  ;zero a register
32D75C          ld   (23767),a
212B29          ld   hl,name            ;get name in hl
22DC5C          ld   (23772),hl         ;store filename
210400          ld   hl,4               ;4=name length
22DA5C          ld   (23770),hl         ;store name length
D9              exx
E5              push hl                 ;preserve hl'
D9              exx
CF              rst  #8
22              defb 34                 ;hook code 34
DD22515C        ld   (23633),ix         ;CURCHL=drive channel
06FF            ld   b,255              ;255 bytes infile
C5          loop: push bc               ;preserve counter
3E00            ld   a,CHAR             ;user would need to
                                        ;insert routine to get
                                        ;character from wherever
                                        ;into a register
D7              rst  #10                ;write to channel
C1              pop  bc                 ;restore counter
10F6            djnz loop
CF              rst  #8
23              defb 35                 ;close file
3E02            ld   a,2
CD0116          call #1601              ;output to screen
D9              exx
E1              pop  hl                 ;restore HL
D9              exx
FB              ei                      ;reenable interrupts
C9              ret
46524544    name: defm "FRED"           ;filename
```

Wenn Sie mehrere Dateien eröffnen, müssen Sie selbstverständlich die Anfangsadressen jedes Kanals in einem sicheren Speicherbereich unterbringen, damit die Dateien auch geschlossen werden können etc. Unsere Routine erzeugt auf der Cartridge eine Datei von 255 Zeichen Umfang.

Auch das Lesen der Daten ist einfach: Die Datei wird wie beschrieben eröffnet und CURCHL auf die Anfangsadresse des Microdrivekanals gesetzt. Danach wird mit der Routine bei # 15E6 (im Haupt-ROM des Spectrum) ein Byte aus dem Kanal gelesen und anschließend im Register A gespeichert.

Hakencode 37 – Wenn IX beim Aufruf diese Codes die Anfangsadresse eines Kanals enthält, liest diese Routine den nächsten Datensatz einer Datei und legt die Daten im Datenbereich des Kanals ab. CHREC wird automatisch inkrementiert, wenn der Lesevorgang erfolgreich war. Die gelesenen Daten überschreiben die Daten des Kanalbuffers.

● **Hakencode 38** – schreibt alle Daten eines Kanals auf Band. IX muß zuvor mit der Anfangsadresse des gewünschten Kanals geladen werden. Beachten Sie, daß jeder Datenbereich, der weniger als 512 Bytex enthält, aber nicht mit einer EOF-Markierung als Dateiende gekennzeichnet ist, vom OS als freier Sektor angesehen wird. Sie sollten daher immer den EOF-Hakencode aufrufen, wenn der letzte Datenblock weniger als 512 Bytes enthält. Nach Aufruf von Hakencode 38 wird der aktuelle Inhalt des Datenbereichs in die Datei geschrieben.

● **Hakencode 39** – liest einen bestimmten Datensatz einer Datei. Dafür wird IX mit der Anfangsadresse des angesprochenen Kanals geladen und CHREC mit der Datensatznummer.

Alles der Reihe nach

Die weiteren Microdriveroutinen lassen sich kaum für die normale Programmierung einsetzen. So liest der Hakencode 40 die Daten eines bestimmten Sektors (in CHREC muß die gewünschte Sektornummer stehen), während Hakencode 41 den nächsten Sektor liest und Hakencode 42 Daten entsprechend in den nächsten freien Sektor schreibt.

Die letzten beiden Hakencodes beeinflussen die Microdrivekanäle. Hakencode 43 definiert einen Kanal und eine Cartridge-Map. In der ersten ROM-Version des Interface 1 war diese Routine jedoch fehlerhaft und löste den Ablauf von Hakencode 34 aus. Hakencode 44 löscht den Kanal, dessen Anfangsadresse in IX steht.

Zum Schluß noch eine Warnung: Setzen Sie die Hakencodes sehr vorsichtig ein! Der Schreibschutz der Microdrives ist softwaregesteuert – ein Programmfehler kann den Inhalt einer gesamten Cartridge zerstören.

Seriell gesteuert

Das Interface 1 des Spectrum bietet Maschinencodeprogrammierern viele interessante Routinen. In diesem Abschnitt untersuchen wir, wie die RS232-Schnittstelle mit „Hakencodes" gesteuert wird.

Sehen wir uns zunächst die neunpolige RS232-Buchse der seriellen Schnittstellen genauer an (unser Bild zeigt die Funktionen der einzelnen Kontakte). Wieviele Leitungen Sie benötigen, hängt im wesentlichen vom Peripheriegerät ab. Beim Betrieb der RS232 müssen jedoch zumindest die Leitungen 2, 3 und 7 verbunden sein und die Leitungen 4 und 5 mit 9 „gebrückt" werden, um auf logisch Eins zu stehen. Die Brücken verhindern, daß sich der Spectrum „aufhängt", wenn das Empfangsgerät am anderen Ende des Kabels nicht bereit ist, Daten entgegenzunehmen. Da dabei Daten verloren gehen, sollten Sie zur Vorbeugung versuchen, immer eine vollgültige RS232-Verbindung herzustellen.

Über die RS232-Schnittstelle lassen sich etwa die verschiedensten Drucker an den Spectrum anschließen. Ist die Kabelverbindung einmal hergestellt, übernimmt die Software die Steuerung. Wenn Sie Text zum Drukker senden wollen, müssen Sie zunächst die Baudrate mit dem BASIC-Befehl FORMAT setzen und dann folgenden Code ausführen:

```
CLOSE #3
OPEN #3;"T"
```

Damit wird Strom 3 dem Kanal T zugeordnet, der den Textkanal der RS232-Schnittstelle darstellt. Die Unterschiede zwischen „Text-" und „Binärströmen" sind im Handbuch des Interface 1 beschrieben. LLIST und LPRINT senden nun Daten zu dem Gerät, das an das Interface 1 angeschlossen ist.

Die Daten werden mit einem Startbit, acht Datenbits und zwei Stopbits übertragen – die entsprechende Formatierung erledigt das Betriebssystem. Wichtiger ist jedoch die Baudrate, also die Sendegeschwindigkeit der Daten. Die Baudrate wird normalerweise mit dem Befehl FORMAT gesetzt, dem aber nur bestimmte Geschwindigkeiten zur Verfügung stehen. Obwohl bei RS232-Schnittstellen nur selten Daten verloren gehen, sollten Sie die Baudrate mit wachsender Entfernung zwischen Sender und Empfänger heruntersetzen.

Bevor Sie die folgenden Techniken einsetzen, müssen Sie die Interfacevariablen mit dem Hakencode 49 anlegen. Die RS232-Schnittstelle wird über zwei Hakencodes und drei Systemvariablen angesprochen. Zunächst die Variablen:

- **BAUD** (bei 23747 und 23748) ist eine Variable im Zwei-Byte-Format, deren Wert die Baudrate für das Senden und Empfangen von Daten bestimmt. Hier stoßen wir jedoch auf eine der Grenzen der seriellen Schnittstelle des Spectrum – es gibt keine Möglichkeit, mit unterschiedlichen Baudraten zu senden und zu empfangen.
- **SERFL** (bei 23751) ist ein Flag-Byte für den Empfang. Wenn es auf Eins steht, ist ein weiteres Byte zum Lesen bereit.
- **SERBT** (bei 23752) ist ein Empfangsbuffer im Ein-Byte-Format. Gelegentlich wird hier auch nach Beendigung der Datenübertragung noch ein Byte empfangen und gespeichert.

Eine vierte Systemvariable – IOBORD (bei 23750) – ist nicht ausschließlich auf die serielle Schnittstelle ausgerichtet. Sie gibt die Farbe des Bildschirmrahmens während der E/A-Vorgänge an. Es ist also möglich, während der Ein- und Ausgabe die Hintergrundfarbe beliebig zu verändern.

Die Baudraten des Interface 1

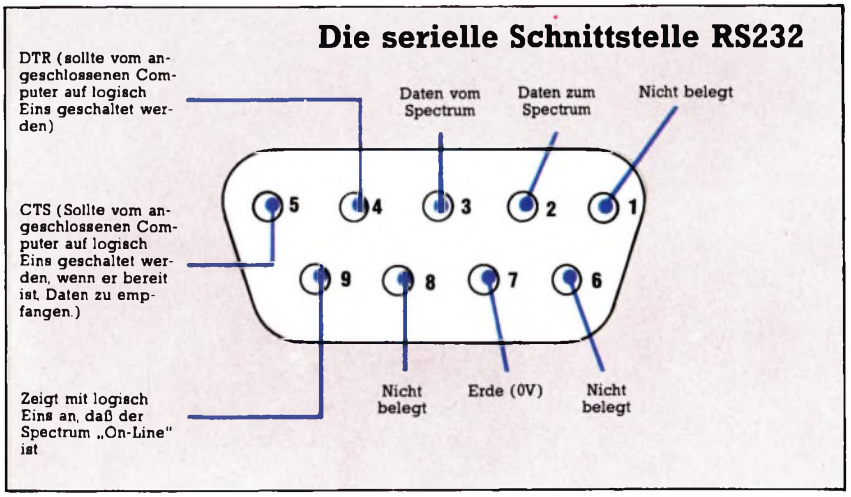
Der RS232-Standard wird hauptsächlich für die Datenübertragung über größere Entfernungen eingesetzt. Die Baudrate gibt an, wieviel Datenbits pro Sekunde übermittelt werden. Die im Artikel angegebene Formel liefert bei den üblichen Baudraten folgende Ergebnisse:

Baudrate	Wert
50	2690
110	1221
300	446
600	222
1200	110
2400	54
4800	28
9600	12
19200	5

Der Wert der gewünschten Baudrate wird mit POKE in die Systemvariable BAUD (bei 23747 und 23748) gesetzt. Da dafür nur eine Variable zur Verfügung steht, gilt für Senden und Empfangen die gleiche Baudrate.

Der folgende Code zeigt, wie die Baudrate von 300 Baud gesetzt wird:

```
LD HL,446
LD (23747),HL
RET
```



Über den seriellen Port des Interface 1 können Daten mit anderen seriellen Geräten ausgetauscht werden. Die Schnittstelle arbeitet mit einer standardmäßigen neunpoligen Buchse.

Für den Einsatz der RS232-Schnittstelle wird zunächst mit dem Befehl FORMAT die Variable BAUD auf einen Wert gesetzt, der sich folgendermaßen errechnet:

Wert = (3500000/26*Baudrate)) −2

Mit dieser Gleichung lassen sich auch nicht standardmäßige Baudraten errechnen, die vom Maschinencode aus eingestellt werden. Außer für die Kommunikation mit einem anderen Spectrum ist das jedoch kaum von Nutzen. Unsere vorseitige Tabelle zeigt Baudraten und die Routine, mit der sie gesetzt werden.

Wie werden nun die Daten über die Leitungen gesandt? In BASIC erzeugt das Eröffnen eines Stroms (OPEN) für die RS232-Schnittstelle einen Kanal mit einer bestimmten Struktur (siehe Bild). Wenn der Kanal indirekt angelegt wurde (z. B. mit SAVE*, MOVE, FORMAT etc.), wird die Zahl 128 zu dem Inhalt von Kanalbyte 4 addiert.

Wenn Sie die RS232-Routinen vom Maschinencode aus einsetzen, brauchen Sie darauf nicht zu achten. Beachten Sie, daß derartigen Kanälen kein Buffer zugeordnet ist. Die Schreib- und Leseroutinen werden über einen Hakencode aufgerufen, der die Adresse in Kanalbyte 5 und 6 (Senden) oder 7 und 8 (Empfangen) anspricht. Hier die beiden RS232-Hakencodes:

- **Hakencode 29** stellt – falls vorhanden – ein Byte der seriellen Schnittstelle in das A-Register. Das Übertragsflag wird in diesem Fall auf Eins gesetzt. Ist kein Byte vorhanden, enthält A den Wert Null, und das Übertragsflag wird entsprechend gelöscht.
- **Hakencode 30** sendet das im Register A gespeicherte Byte zur Datenübertragung an die serielle Schnittstelle. Beim Einsatz dieses Codes gibt es auf Maschinencodeebene keinen Unterschied zwischen Strom B und T, da alle Bytes unverändert übertragen werden.

An dieser Stelle können jedoch Probleme mit SERBYT und SERFL auftreten. Wenn die Hakencoderoutine feststellt, daß SERFL auf Eins steht, holt sie ein Byte von SERBYT, selbst wenn die serielle Kommunikation bereits beendet wurde und das Byte daher nicht mehr gültig ist. Dieses „Fehlerbyte" muß abgefangen werden, da es die Bedeutung der darauffolgenden Bytes verändert, sobald es sich am Anfang von Empfangsdaten befindet. Wenn Sie SERFL vor jedem Lesevorgang (aus der seriellen Schnittstelle) auf Null setzen, wird der Inhalt von SERBYT ignoriert. Hier der entsprechende Code:

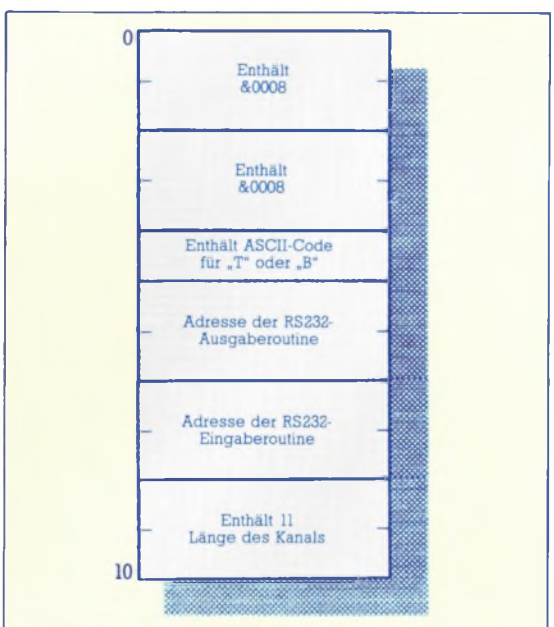
```
XOR A
LD   (23751),A
```

Hier liegt eigentlich die einzige Schwierigkeit, die beim seriellen Interface auftreten kann. Wenn Sie überhaupt keine Daten erhalten, sollten Sie die Leitungen im Stecker vertauschen, die zu Kontakt 2 und 3 führen – oft liegt hier der Fehler. Die erzeugte Baudrate ist zwar nicht immer exakt, liegt aber im Rahmen des RS232-Standards.

Der folgende Code überträgt 255 Bytes an die RS232-Schnittstelle:

```
          ;send 255 bytes of data down serial interface
          ;
CF                rst  #8              ;set up...
31                defb 49              ;...I/face I variables
216E00            ld   hl,110          ;value for 1200 Baud
22C35C            ld   (23747),hl      ;set baud rate
06FF              ld   b,255           ;set counter
C5        loop:   push bc              ;save counter
3E00              ld   a,DATA          ;user supplies routine..
CF                rst  #8              ;...to get data in a reg
1E                defb 30              ;send the byte in a
C1                pop  bc              ;restore counter
10F8              djnz loop
C9                ret
```

In vielerlei Hinsicht ist die RS232-Verbindung die einfachste Funktion des Interface 1. Zwei Routinen für Empfangen und Senden erledigen die Arbeit, ohne daß der Maschinencode Kanalinformationen steuern muß.



Die Struktur des RS232-Kanals
Beachten Sie, daß diesem Kanal kein Buffer zugeordnet ist. Die Daten werden mit den Hakencodes 29 und 30 über das A-Register geleitet.

Spectrum am Netz

Das Interface 1 des Spectrum wird hauptsächlich zur Steuerung der Microdrives oder als serielle Schnittstelle RS232 eingesetzt. Es kann Ihnen aber auch Zugang zu lokalen Netzwerken (LAN) verschaffen.

Wir beschrieben bereits, wie das Interface 1 die Microdrives und die serielle Schnittstelle steuert. In diesem Artikel wollen wir untersuchen, wie sich die Netzwerkfähigkeiten des Interface 1 einsetzen lassen. Die Arbeitsweise von Netzwerken wurde in einem früheren Teil des Kurses schon genauer erklärt. Im folgenden Text bezeichnet SELF unsere Maschine und IRIS die Maschine im Netzwerk, mit der wir kommunizieren wollen.

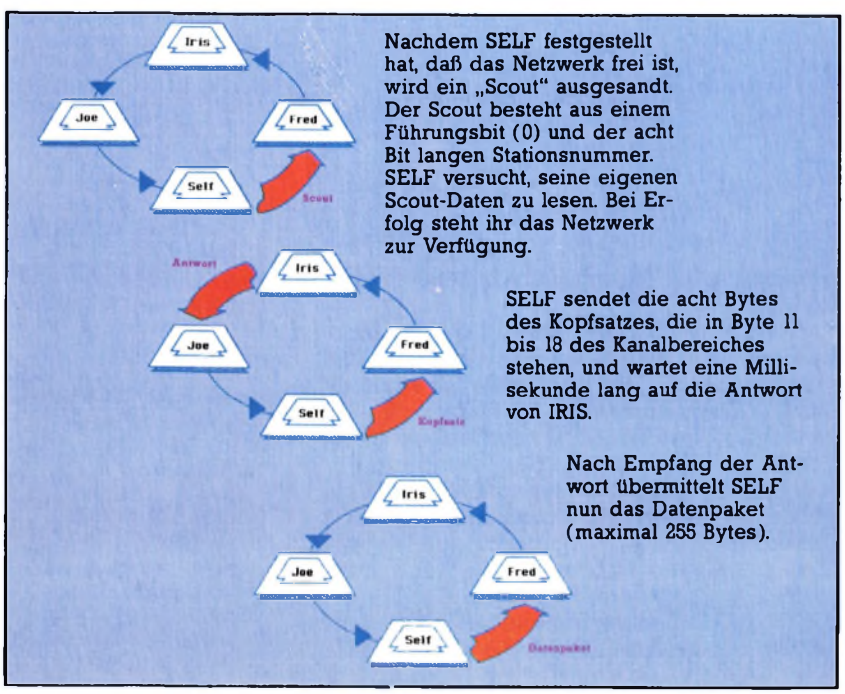
Programme und Daten lassen sich mit zwei verschiedenen Techniken über ein Netzwerk senden. Beim „Broadcasting" kann jede Maschine des Netzwerks „mithören". Dieses Verfahren verteilt die Daten schnell an viele Maschinen. Bei der „Exklusivübertragung" gibt SELF an, daß nur IRIS die Information empfangen soll. Wenn IRIS keine Daten entgegennimmt, versucht SELF mehrfach eine Kommunikation herzustellen.

Auch die Abläufe, mit denen die Maschinen eines Netzwerks untereinander Daten austauschen, sind interessant. Ein Gerät, das Daten senden will, prüft zuerst, ob das Netzwerk frei ist. Damit nicht alle Maschinen gleichzeitig zu senden beginnen (und dadurch ihre Daten vermischen), wartet jede Maschine eine zufallsgesteuerte Zeit, nachdem sie festgestellt hat, daß das Netz frei ist, und nimmt erst dann das Netz für sich in Anspruch. Wenn es zu diesem Zeitpunkt bereits wieder aktiv ist, muß das Gerät nochmals warten.

In das freie Netz sendet SELF zunächst einen „Scout". Der Scout enthält die Stationsnummer der Maschine, zu der SELF Verbindung aufnehmen will. Ist sie damit erfolgreich, hat SELF das Netz „unter Kontrolle".

Nun wird ein „Datenpaket" gesandt, das aus einem Kopfsatz (mit Informationen über den darauffolgenden Datenblock) und 255 Datenbytes besteht. Der nächste Vorgang hängt von der Art der Übertragung ab. Bei einer Exklusivübertragung wartet SELF eine Millisekunde lang auf das Antwortsignal der angesprochenen IRIS. Trifft dieses Signal nicht ein, sendet SELF acht Millisekunden später nochmals Scout und Kopfsatz. Nur bei Empfang einer Antwort wird auch der Datenblock abgesandt. Eine Broadcastübertragung wartet dagegen nicht erst auf eine Antwort, sondern speist den Datenblock sofort ins Netz ein.

Ein Netzvorgang „belegt" zuerst das Netz für die Sendestation und übermittelt dann die Daten (angeführt von einem Kopfsatz). Das Bild zeigt, wie in einem Netzwerk mit vier Geräten die Station SELF Daten exklusiv an die Station IRIS übermittelt.



Die Angriffspunkte

Vom Maschinencode aus stehen vier Hakencodes und zwei wichtige Systemvariablen für die Netzsteuerung zur Verfügung.

- **NTSTAT** (bei 23749) enthält die Stationsnummer der Maschine. Wenn der Befehl FORMAT in einem Netzwerk verwandt wird, setzt er diese Variable. Ihr Standardwert ist Eins.
- **DSTR1**: Diese Variable kennen Sie bereits von den Microdrives. Sie liegt bei 23766 und 23767 und enthält die Stationsnummer der IRIS, an die SELF Daten senden will. Wenn die Variable auf Null steht, werden alle Daten per Broadcasting übertragen.

Die Hakencodes für das Netzwerk funktionieren ähnlich wie die der Microdrives. Zunächst wird mit Hakencode 45 ein Kanal mit 276 Bytes angelegt (das Bild zeigt seine Struktur), der sich wie jeder andere Kanal im Speicherbereich für Kanaldaten befindet.

Der für das Netzwerk bestimmte Kopfsatz enthält die Bytes von NCIRIS bis einschließlich NCHCS. Der Datenblock reicht von Kanalbyte 21 bis 275. Die Kanalbytes 19 und 20 haben nur Bedeutung, wenn über den Kanal Daten aus dem Netzwerk gelesen werden. Wie beim Microdrivesystem läßt sich leicht in den Kanal schreiben oder daraus lesen. Dabei wird die Kanaladresse an CURCHL übergeben und

RST # 10 oder CALL # 15E6 aufgerufen. Bei einem Netzwerk löst man jedoch automatisch eine Übertragung von Kopfsatz und Daten aus, wenn versucht wird, das 256te Byte in den Datenbereich zu schreiben.

● **Hakencode 45** eröffnet einen Netzwerkkanal. Für seinen Einsatz wird NISTAT auf die Stationsnummer von SELF gesetzt und DSTR1 auf die von IRIS (beim Lesen werden die Daten von IRIS an SELF übermittelt). Sie sollten auch hier sicherstellen, daß die Systemvariablen des Interface 1 gesetzt sind. Der Hakencode wird auf die übliche Art eingesetzt. Beim Rücksprung enthält das IX-Register die Anfangsadresse des Kanals.

Wie bereits erwähnt, sendet RST # 10 Bytes über das Netzwerk, und CALL # 15E6 empfängt Daten. Dabei kann jedoch der Inhalt des IX-Registers zerstört werden. Sie sollten daher vor Aufruf dieser Routinen den Inhalt von IX sichern, da Sie ihn – oder zumindest die Kanaladresse – für das Schließen des Kanals unbedingt benötigen.

● **Hakencode 46** schließt einen Netzwerkkanal. Bei Aufruf der Routine muß sich die Anfangsadresse des Kanals im IX-Register befinden. Wenn der Kanal zum Schreiben eröffnet war, werden alle Daten, die sich noch im Datenbereich des Kanals befinden, in das Netzwerk übertragen. Nach Schließen des Kanals verschiebt sich der Speicher zwischen Kanalende und STKEND nach unten.

● **Hakencode 47** kann ein bestimmtes Paket aus dem Netzwerk empfangen. Die Kanalvariable NCNUMB muß dabei die gewünschte Blocknummer enthalten. Es wird vorausgesetzt, daß auch in NCSELF und NCIRIS die korrekten Werte gespeichert sind (die normalerweise gleich bei Eröffnung des Kanals gesetzt werden). Die Hakencoderoutine versucht nun, das Paket zu empfangen. Wenn ein „Time Out" eingetreten ist, die Daten nicht sauber empfangen wurden, oder die Prüfsummen nicht übereinstimmen, steht das C-Flag nach dem Ablauf auf Eins. Anhand des Status der Flags läßt sich feststellen, ob Sie ein bestimmtes Paket nochmals aus dem Netzwerk anfordern müssen. Ist das Paket „unbeschädigt" eingetroffen, wird – falls nötig – eine Bestätigung gesandt und NCNUMB inkrementiert.

● **Hakencode 48** übermittelt ein Datenpaket über das Netzwerk und ist damit die wichtigste Übertragungsroutine. Der Code arbeitet mit einem offenen Kanal, dessen Adresse per IX-Register übergeben wird. Register A gibt den Dateityp des im Kanal gespeicherten Datenblocks an. Wenn das Register A auf Eins steht, enthält der ausgesandte Block das Dateiende. Für alle normalen Datenblöcke ist der Registerwert Null.

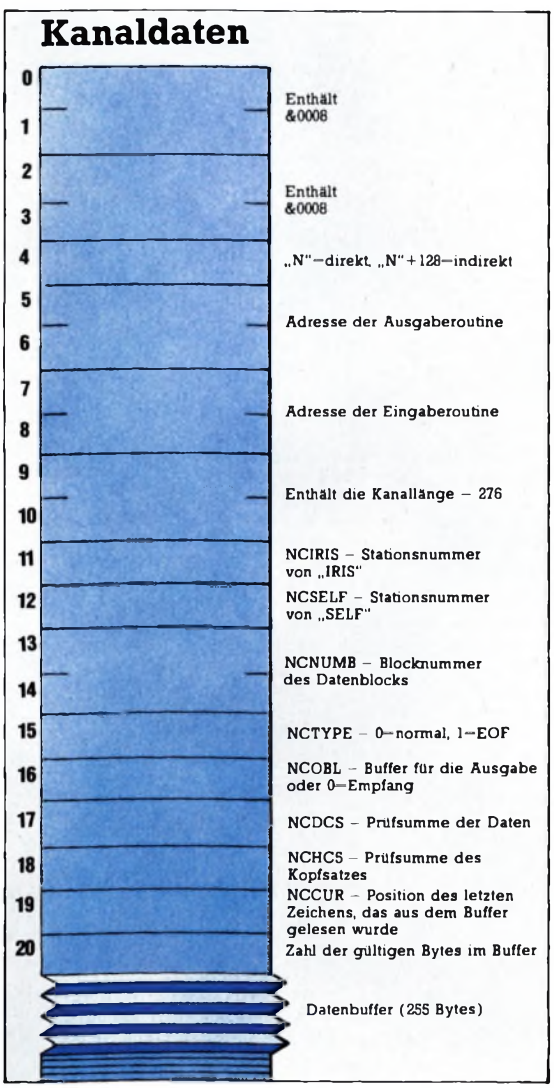
Beim Öffnen eines Kanals bestimmt der Wert von DSTR1, ob die Daten per Broadcast oder per Exklusivübertragung gesandt werden. Wenn die Empfangstation die Daten angenommen hat, wird zunächst der Kopfsatz in den Kopfbereich des Kanals gestellt. NCOBL enthält die Anzahl der übermittelten Bytes und NCTYPE die Information, ob das empfangene Paket das Dateiende markiert.

Bedarfsgerecht

Bei einer erfolgreichen Übertragung erhöht sich die Kanalvariable NCNUMB um Eins.

In einem LAN-Netzwerk können Sie die Übermittlung Ihrer Daten mit diesen zwei Methoden genau auf Ihren Bedarf einstellen. In einem kleinen Netz mit nur zwei Maschinen sind die Hakencodes 48 und 47 allerdings weniger geeignet als die einfache Kommunikation mit RST # 10 und CALL # 15E6.

Damit endet unsere Untersuchung der Datenübertragungsmöglichkeiten des Interface 1. In der nächsten Folge sehen wir uns die neuen BASIC-Befehle des Interface 1 an. Darüberhinaus machen wir Sie damit vertraut, wie Sie mit Hilfe des Interface 1 eigene BASIC-Befehle definieren können, damit das BASIC Ihres Spectrum noch leistungsfähiger wird und Ihre eigenen Programme noch professioneller.



Hakencode 45 legt einen Netzwerkkanal von 276 Bytes an, in dem 20 Bytes für die Informationen des Kopfsatzes und 255 Bytes als Datenbuffer zur Verfügung stehen. Das Bild zeigt die Bytestruktur des Kopfsatzes.

Erhöhte Befehlsgewalt

Mit dem Interface 1 können Sie eigene Befehle in das BASIC des Spectrum einbauen. Diese Möglichkeit ist außerordentlich nützlich, da das Interface 1 die Microdrives und andere Geräte nur mit neuen Befehlen steuern kann. Wir untersuchen die Einbautechnik.

Um neue BASIC-Befehle einsetzen zu können, müssen wir zunächst einmal die Fehlerbehandlungsroutinen des Spectrum „täuschen", damit sie die neu definierten Befehle nicht zurückweisen. Wir beschäftigen uns daher zuerst ausführlich mit den Fehlerroutinen des Sinclair Spectrum.

Sofort nach einem Druck auf die ENTER-Taste wird jede neu eingegebene BASIC-Zeile vom BASIC-Interpreter auf Fehler überprüft. Diese Prüfung fängt fehlende Variablen, falsch eingesetzte Parameter etc. ab, damit der Spectrum nicht von groben Eingabefehlern „belästigt" wird. Zeilen ohne offensichtliche Fehler werden entweder an die angegebene Programmposition gesetzt oder, sofern es sich um direkte Befehle handelt, sofort ausgeführt.

Enthält die Zeile Fehler, wird der Befehl RST#8 ausgeführt, gefolgt von DEFB, der dem Betriebssystem die Art des Fehlers anzeigt. Das OS zeigt nun das blinkende Fragezeichen.

Nach dem ersten Test wird die Syntax der Zeile nochmals überprüft. Dieser zweite Test bewertet beispielsweise die angegebenen numerischen Ausdrücke und ruft dann die entsprechenden ROM-Routinen auf. In diesem Stadium werden Fehler wie „Variable existiert nicht" entdeckt.

Der erste Test (auch Syntaxprüfung genannt) stellt sicher, daß alle Anweisungen richtig geschrieben wurden, führt den Befehl aber nicht aus. Erst der zweite Test – Ablaufprüfung genannt – aktiviert den Befehl. Mit dem Einbau des Interface 1 (IF1) ändert sich dieser Ablauf jedoch. Wir hatten bereits gesehen, daß das IF1 beim Ansprechen der Adressen #08 oder #1708 das Haupt-ROM ausblendet und statt dessen das Schatten-ROM aktiviert. Der Spectrum arbeitet nun solange mit dem Schatten-ROM, bis das Programm die Adresse #700 des IF1 anspricht und damit das Haupt-ROM wieder anschaltet.

Ein Fehler aktiviert also sofort das Schatten-ROM, das eine weitere Fehlerprüfung durchführt. Wenn der Fehler durch einen der legalen Interface-1-Befehle wie CAT oder FORMAT verursacht wurde, führt das ROM des Interface 1 die entsprechenden Routinen aus und übergibt die Steuerung ans Haupt-ROM.

Wenn die Fehlerbedingung jedoch nicht von einem der Interface-1-Befehle verursacht wurde, verzweigt die Routine zu der in der Systemvariablen VECTOR (im IF1-ROM bei 23735 und 23736) gespeicherten Adresse. VECTOR enthält die Anfangsadresse der Fehlerbearbeitungsroutine des Schatten-ROM. Beachten Sie, daß diese Adresse bei den unterschiedlichen ROM-Versionen des IF1 an einer anderen Position liegt. In der nächsten Folge gehen wir auf die wichtigsten Unterschiede dieser Versionen genauer ein und erklären, wie man die Versionsnummer feststellt.

Die erste ROM-Version des Interface 1 arbeitet mit VECTOR = #1F0. Dabei führt ein Syntaxfehler direkt zurück zum Editor. Wenn Sie nun die in VECTOR gespeicherte Adresse ändern, können Sie die Fehlerbehandlung auf Ihre eigene Routine leiten.

#0010

Dieses Modul ist die einzige Möglichkeit, vom Schatten-ROM aus Routinen des Haupt-ROMs aufzurufen. Das Modul führt das An- und Abschalten des Haupt-ROMs automatisch aus. Es wird folgendermaßen eingesetzt:

```
RST     #0010
DEFW    adresse     ;Adresse der aufgerufenen Routine
```

#0020

Diese Routine des Schatten-ROM entspricht dem RST #8 des Haupt-ROM. RST #0020, ge-

Schattenspiele

Das Schatten-ROM des Interface 1 besitzt außer der Steuerung der Microdrives, der RS232-Schnittstelle und der Netzwerkeigenschaften auch eine Reihe nützlicher Routinen, darunter ein Dezimal-Hexadezimalwandler.

Das Buch „Shadow ROM Disassembly" von Melbourne House (ISBN 0–86161–191–8) enthält die ROM-Routinen in allen Einzelheiten. Zusätzlich zeigt es einige praktische Dienstprogramme.

folgt von einem Byte mit dem passenden Fehlercode, aktiviert das Modul. Unsere Tabelle zeigt die möglichen Fehlermeldungen.

#0028

Wird zur Erzeugung einer „normalen" Fehlermeldung über das Haupt-ROM, nicht mit dem Schatten-ROM, eingesetzt. Vor Aufruf dieser Routine muß die Adresse 23610 mit dem entsprechenden Fehlercode geladen werden.

#01F0

Stellt während der Syntaxprüfung einen blinkenden Cursor dar.

#05B7

Der Aufruf dieser Routine zeigt an, daß die Syntaxprüfung abgeschlossen ist.

#05C1

Der Befehl beendet die Befehlsausführung des Schatten-ROM und gibt die Steuerung an das Haupt-ROM zurück.

#0700

Veranlaßt die Rückkehr zum Haupt-ROM. Es ist eine Möglichkeit, das Schatten-ROM zu verlassen und das Haupt-ROM zu aktivieren. Zum Schluß noch den entsprechenden Hakencode:

● **Hakencode 50**

sollte – wie alle anderen Hakencodes – nicht in Verbindung mit dem Schatten-ROM eingesetzt werden. Der Code ermöglicht, vom Haupt-ROM aus Routinen des Schatten-ROM anzusprechen, und sollte nur eingesetzt werden, wenn die Adressen des Schatten-ROM genau bekannt sind. Beim Aufruf des Moduls muß die Adresse der gewünschten Schatten-ROM-Routine im Register HL gespeichert sein.

```
LD HK,adresse; Adresse der Routine
LD     (23789),HL
RST    #8
DEFB   50
```

Bei einem aktiven Schatten-ROM sind auch die „Restart"-Befehle des Z80 umgestellt. Außerdem wird die Tastatur nicht mehr abgefragt und die Systemvariable FRAMES nicht inkrementiert. Eine längere Aktivierung des Schatten-ROM stört daher auch die Zeitmessung, die diese Variable steuert.

Gelegentlich gibt es auch Probleme beim Einsatz der Fließkommaberechnung vom Schatten-ROM aus. Da die Routine bei #0010 Schwierigkeiten mit dem Aufruf dieses Moduls zu haben scheint, ist es besser, dazu das Schatten-ROM zu verlassen und mit dem Haupt-ROM zu arbeiten. Das Schatten-ROM können Sie mit dem Hakencode 50 aufrufen.

Byte	Fehlermeldung	Byte	Fehlermeldung
00	Ungültiger BASIC-Befehl	11	Strom bereits eröffnet
01	Ungültige Stromnummer	12	Schreiben in Lesedatei
02	Ungültige Gerätebezeichnung	13	Lesen aus Schreibdatei
03	Ungültiger Name	14	Laufwerk schreibgeschützt
04	Ungültige Laufwerksnummer	15	Microdrive voll
05	Ungültige Stationsnummer	16	Microdrive nicht angeschlossen
06	Fehlender Name	17	Datei nicht gefunden
07	Fehlende Stationsnummer	18	Fehler im Hakencode
08	Fehlende Laufwerksnummer	19	CODE-Fehler
09	Fehlende Baudrate	20	MERGE-Fehler
10	Kopfsatz stimmt nicht überein	21	Verifikation nicht erfolgreich
		22	Falscher Dateityp
		255	Programm beendet

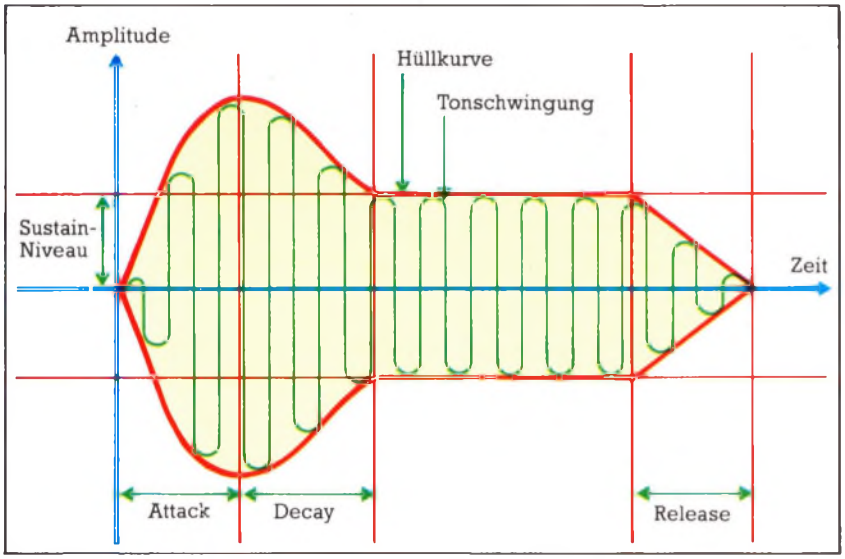
Einblick mit PEEK

Mit diesem Programm können Sie die Routinen des Schatten-ROM selbst untersuchen. Aus der Disassemblierung fremder Programme kann man einiges lernen – unter anderem den optimalen Einsatz der untersuchten Routine. Leider können Sie das Schatten-ROM nicht direkt mit PEEK ansprechen. Das folgende kurze BASIC-Programm lädt daher einen beliebigen Abschnitt des Schatten-ROM in den Arbeitsspeicher, wo Sie es bequem untersuchen können.

```
10   GO SUB 200
20   INPUT "How many bytes do you want to
     CLEAR?";b
30   CLEAR r-(b+24)
40   GO SUB 200
50   FOR n=1 TO 23
60   READ d: POKE r+n,d
70   NEXT n
80   INPUT "Start address in Shadow ROM?";s
90   INPUT "Number of bytes to copy?";b
100  LET z=s: GO SUB 300
110  POKE r+12,l: POKE r+13,h
120  LET z=b: GO SUB 300
130  POKE r+18,l: POKE r+19,h
140  LET z=r+9: GO SUB 300
150  POKE r+2,l: POKE r+3,h
160  LET z=r+24: GO SUB 300
170  POKE r+15,l: POKE r+16,h
180  RANDOMIZE USR (r+1)
190  PRINT "Data stored at" :r+24
195  STOP
200  LET r=PEEK 23730+256*PEEK 23731:
     RETURN
300  LET h=INT (z/256): LET l=z-256*h: RETURN
400  DATA 33,0,0,34,237,92,207,50,
     225,225,33,0,0,17,0,0,1,0,0,237,176,
     199,201,
```

Trommelwirbel

Der SID-Chip (Sound Interface Device) erzeugt die Klänge des Commodore 64 und ist eigentlich ein eigener Synthesizer, der mit entsprechender Software sogar roboterähnliche Sprache erzeugen kann und über enorme Klangfähigkeiten verfügt.



Der Charakter eines Tones – das heißt die Gruppe von Merkmalen, mit der wir beispielsweise zwischen einem Klavier und einer Violine unterscheiden – hängt von der Form der Hüllkurve ab. In der elektronischen Klangsynthese besteht eine Hüllkurve aus vier klar voneinander getrennten Abschnitten – Attack, Decay, Sustain und Release (ADSR). Die Dauer der einzelnen ADSR-Abschnitte wird mit POKE-Werten in die Register des SID-Chips geladen.

Die vom Commdore 64 erzeugten Töne werden normalerweise über die RF-Buchse direkt zum Fernseher geleitet. Die Klangausgabe läßt sich jedoch sehr verbessern, wenn Sie an den Audio/Video-Ausgang ein HiFi-System anschließen. In dieser Folge untersuchen wir, wie sich der Commodore 64 per Software in eine Rhythmusmaschine verwandeln läßt. Bei Anschluß eines HiFi-Systems können dabei erstaunlich realistische Töne entstehen.

Der SID-Chip kann jedoch nicht nur Töne erzeugen, sondern auch Audiosignale entgegennehmen, die von externen elektronischen Geräten oder von Musikinstrumenten (z. B. elektrischen Gitarren) stammen. Das externe Signal läßt sich mit der Audio-Ausgabe des SID-Chips mischen. Seien Sie beim Einsatz von E/A-Schnittstellen jedoch vorsichtig, da falsch angeschlossene Leitungen den Computer beschädigen können. Schlagen Sie vor dem Anschließen im Hardwarehandbuch die korrekten Verbindungen und Voltzahlen nach.

Wir untersuchen zunächst, wie sich Musikklänge aus einzelnen, reinen Tönen zusammensetzen. Danach beschreiben wir eine andere Methode, die die gleiche Wirkung erzielt: Über die Hüllkurve eines Tonablaufs lassen sich Klänge mit unterschiedlichem Charakter erzeugen. Das bedeutet, daß man durch Ändern der Steuerwerte fast jeden Klang hervorrufen kann. Die Steuerwerte heißen auch „ADSR" (Attack, Decay, Sustain und Release).

Leider eignet sich das BASIC des Commodore 64 nicht besonders zur Steuerung von Klängen. Eine wirksame Tonerzeugung kann daher nur mit vielen PEEK- und POKE-Befehlen erreicht werden.

Klänge erreichen unser Ohr in Form von periodischen Luftschwingungen, wobei die Zahl der Schwingungen pro Sekunde „Tonhöhe" oder „Tonfrequenz" genannt wird. Die untere Hörschwelle des menschlichen Ohrs liegt zwar bei 15 Zyklen pro Sekunde (15 Hertz), doch empfinden wir einen reinen Ton von 100 Hertz noch als tief; der Kammerton A über dem mittleren C liegt bei 440 Hz. Durch Frequenzverdoppelung steigt die Tonhöhe um eine Oktave. Der Mensch kann normalerweise etwa 10,5 Oktaven wahrnehmen, von denen der dreistimmige SID-Oszillator etwa acht erzeugen kann – von 0 bis 4000 Hz.

Der französische Physiker Jean Fourier (1768 bis 1830) entdeckte als erster, daß jede periodische Wellenform aus einer reinen Grundschwingung und Obertönen besteht. Die Obertonfrequenzen sind ganzzahlige Vielfache der Grundschwingung und werden „Harmonische" genannt. Der charakteristische Klang eines Instrumentes wird vom Lautstärkenverhältnis der Obertöne (oder „Harmonischen") zueinander bestimmt.

Reine Töne entsprechen reinen Sinusschwingungen, die sich – als Analogsignal – nicht leicht auf digitalen Geräten mit nur 0 oder 5 Volt erzeugen lassen. Statt reine Frequenzen zu generieren und sie zu periodischen Klängen zusammenzumischen, arbeiten Heimcomputer daher mit einer anderen Technik.

Schwingungsformen

Jede der drei Stimmen des SID-Chips kann je nach Einstellung eine von vier periodischen Schwingungsformen erzeugen:

1) **Die Sägezahnschwingung** enthält alle Harmonischen. Die Intensität der Nten Harmonischen entspricht proportional 1/N.

2) **Die Dreieckschwingung** enthält nur die ungeraden Harmonischen. Die Intensität der Harmonischen N entspricht hier proportional $1/N^2$.

3) **Eine Rechteckschwingung** enthält ungerade Harmonische proportional zu 1/N. Durch Änderung der „Impulsbreite" lassen sich unterschiedliche Rechteckwellen mit eigener har-

monischer Mischung erzeugen.
4) **Das weiße Rauschen** ist eine zufällige Frequenzmischung, die oft für Spezialeffekte eingesetzt wird.

Der harmonische Inhalt kann weiterhin noch durch die drei Filter des SID-Chip (Low-Pass, Band-Pass und High-Pass) verändert werden. Ein Low-Pass-Filter läßt beispielsweise alle Frequenzen unter einem bestimmten Wert passieren, schwächt aber alle höheren Frequenzen ab. Mit diesen Möglichkeiten und der Steuerung der ADSR-Hüllkurve läßt sich fast jeder Klang erzeugen.

Auch die Sprachsynthese kann mittels mehrerer Techniken erzeugt werden. Dabei werden die Grundelemente der Sprache – sogenannte „Phoneme" – in einem Array von ADSR-Werten gespeichert. Da die deutsche Sprache nur etwa 52 verschiedene Phoneme besitzt, ist die Codierung nicht zu umfangreich. Mit einem Maschinencodeprogramm werden die ASC-Texte (die Commodoreversion von ASCII) in eine Folge von Phonemcodes übersetzt und dann über das ADSR-Array an den SID-Chip gesandt. Das hört sich einfacher an, als es ist, da die Regeln zur Übersetzung von Text in Phoneme recht kompliziert sind. Die Sprachqualität hängt wesentlich von der Arbeit dieses Programmteils ab. Da sich die Sprachsynthese auf dem Commodore 64 gut einsetzen läßt, gibt es schon eine Reihe kommerzieller Produkte, die mit dieser Technik arbeiten.

Klangentwicklung

Unser ADSR-Diagramm zeigt die allgemeine Form eines Klanges und betont dabei die Teile, die der SID-Chip über ein Programm steuern kann. Die vier ADSR-Faktoren sind:
1) **Attack:** Die Anschlagslautstärke
2) **Decay:** Das Abklingen eines Tons auf ein konstantes Niveau
3) **Sustain:** Die Lautstärke auf konstantem Niveau
4) **Release:** Das Ausklingen des Tones vom Sustainniveau auf Null.

Die Abschnitte von Attack, Decay und Release werden mit Nybbles (vier Bits) der SID-Register gesteuert. Jeder Parameter kann daher Werte von 0 bis 15 annehmen, die mit POKE in die SID-Register gestellt werden. Unsere Tabelle links unten veranschaulicht die Dauer der Abläufe.

Die Dauer des Sustain-Abschnitts wird durch eine Verzögerungsschleife abhängig vom Notenwert bestimmt. In unserer zweiten Tabelle haben wir die für eine Violine typischen ADR-Werte aufgeführt. Der Commodore 64 erzeugt seine Klänge in sechs Schritten.

Wert	Attack-Rate (Zeit/Zyklus)	Decay/Release-Rate (Zeit/Zyklus)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 secs
11	800 ms	2.4 secs
12	1 sec	3 secs
13	3 secs	9 secs
14	5 secs	15 secs
15	8 secs	24 secs

ADSR	Zeit	POKE-Wert
A	500ms	10
D	300ms	8
S	—	—
R	750ms	9

Hier ein Beispiel, bei dem SID=54272 die Basisadresse des Chips ist.

- Schritt 1: Lautstärke einschalten

```
POKE SID+24,15
```

- Schritt 2: ADSR auswählen, z. B.:

```
POKE SID+5,9 :REM ATTACK/DECAY
  STIMME #1
POKE SID+6,0 :REM SUSTAIN/RELEASE
  STIMME #1
```

- Schritt 3: Für jeden Oszillator die Frequenz wählen, z. B.:

```
POKE SID+1,25 :REM HI-BYTE FREQUENZ
  STIMME #1
POKE SID,0 :REM LO-BYTE FREQUENZ
  STIMME #1
```

- Schritt 4: Wellenform wählen, z. B.:

```
POKE SID+4,33 :REM SÄGEZAHN FÜR
  STIMME #1
```

An diesem Punkt fängt der Ton an zu klingen – im Fachjargon: „Es öffnet sich das Gatter".

- Schritt 5: Eine Verzögerungsschleife für die Dauer des Tons auf der Sustain-Ebene.

- Schritt 6: Die Wellenform muß freigegeben werden, z. B.:

```
POKE SID+4,32 :REM SÄGEZAHN FREI-
  GEBEN
```

Eine Melodie können Sie auf dem Commodore 64 am einfachsten spielen, wenn Sie die Anfangswerte von ADSR setzen und mit einer FOR . . .NEXT-Schleife die hi/lo-Bytes der Frequenzen mit READ aus DATA-Anweisungen lesen. Wenn Sie in die DATA-Befehle Nullen einsetzen, verändert sich das Tempo der verschiedenen Stimmen, während Delay hingegen konstant bleibt.

Trommelfeuer

Unser Rhythmusprogramm arbeitet mit einem IRQ-Keil, der vom BASIC-Programm unabhängig ist. Dabei prüft der Maschinencode über den Inhalt der Speicherstelle 197 ($00C5), welche Taste gedrückt wurde. In dieser Zero-Page-Adresse befindet sich immer das zuletzt eingegebene Zeichen. Wenn $00C5 oft genug abgefragt wird, brauchen Sie die Betriebssystemroutine zur Tastaturabfrage nicht vom Programm aus extra aufzurufen.

Da der BASIC-Treiber nicht wie GET oder INPUT den Tastaturbuffer zurücksetzt, muß das Programm vor dem Rücksprung den Pointer in Adresse 198 ($00C6) auf Null stellen. Diese Adresse der Zero-Page enthält normalerweise die Zahl der im Tastaturbuffer gespeicherten Zeichen.

Das Programm ordnet jedem Schlaginstrument 16 Takteinheiten zu, mit denen sich leicht ein Rockrhythmus im Vierertakt anlegen läßt. Die Takteinheiten werden in den einzelnen Feldern eines Bildschirmrasters angezeigt und lassen sich mit dem Cursor und der Return-Taste aktivieren oder deaktivieren. Die Taste f1 startet schließlich den Rhythmus. Da die Klänge interruptgesteuert sind, kann das Rhythmusmuster editiert werden, während der Rhythmus weiterläuft.

Das BASIC-Programm enthält den gesamten Maschinencode in Form von DATA-Befehlen. Wir haben jedoch auch den Quellentext des Maschinencodes abgedruckt. Wenn Sie damit arbeiten, sollten Sie im BASIC-Programm allerdings die Zeilen 1520 bis 1540 und die DATA-Anweisungen zwischen Zeile 1620 und 1940 löschen.

Probieren Sie nach der Programmeingabe doch einmal folgenden klassischen Rhythmus:

Das Rhythmusprogramm in BASIC

```
1000 REM **RHYTHM GENERATOR**
1010 PRINTCHR$(147):REM CLEAR SCREEN
1020 GOSUB1510:REM SETUP M/C+SPRITE
1030 GOSUB1270:REM SETUP SCREEN
1040 SYS(49152):REM *INSERT WEDGE*
1050 REM SYS(49175) TO REMOVE WEDGE
1060 REM *MAIN LOOP*
1070 P=PEEK(197):REM CURRENT KEYPRESS
1080 IFP=37THENY=Y-16:SY=SY-1:IFY<95THENY=95:SY=0
1090 IFP=36THENY=Y+16:SY=SY+1:IFY>127THENY=127:SY=
2
1100 IFP=47THENX=X-16:SX=SX-1:IFX<28THENX=28:SX=0
1110 IFP=44THENX=X+16:SX=SX+1:IFX>268THENX=268:SX=
15
1120 Z=0:IFP=1THENZ=1
1130 PRINTCHR$(19);S$;"TEMPO =";PEEK(679);CHR$(157
);"  "
1140 POKEVIC+16,X/256:POKEVIC,XAND255
1150 POKEVIC+1,Y
1160 REM**COMPUTE SCREEN LOC FROM SPRITE X,Y POS
1170 SC=B+INT((Y-50)/8+1)*40+INT((X-18)/8)
1180 IFSY=0THENROW=50000:REM BASS DRUM
1190 IFSY=1THENROW=50016:REM SNARE DRUM
1200 IFSY=2THENROW=50032:REM COW BELL
1210 IFZ=1ANDPEEK(SC)=32THENPOKESC,24:POKEROW+SX,1
:GOTO1230
1220 IFZ=1ANDPEEK(SC)=24THENPOKESC,32:POKEROW+SX,0
1230 IFP=51THENGOSUB1270:FORI=0TO50:POKE50000+I,0:
NEXT
1240 IFP=57THENPOKE198,0:POKEVIC+21,0:PRINTCHR$(14
7):END
1250 GOTO1070
1260 REM *SET UP SCREEN*
1270 PRINTCHR$(19);
1280 S$=CHR$(17)+CHR$(17)+CHR$(17)
1290 PRINTTAB(3)"
"
1300 PRINTCHR$(18)TAB(3)" **** CBM64  RHYTHM GENER
ATOR **** "
1310 PRINT:PRINT:PRINT
1320 PRINT" - - - - - - - - - - - - - - - - "
1330 PRINT"                                -BASS"
"
1340 PRINT" - - - - - - - - - - - - - - - - "
1350 PRINT"                                -SNARE
"
1360 PRINT" - - - - - - - - - - - - - - - - "
1370 PRINT"                                -BELL"

1380 PRINT" - - - - - - - - - - - - - - - - "
1390 PRINT:PRINTTAB(8)"OPTIONS...."
1400 PRINTTAB(8)"[F1]    PLAY RHYTHM"
1410 PRINTTAB(8)"[F3]    DECREASE TEMPO"
1420 PRINTTAB(8)"[F5]    INCREASE TEMPO"
1430 PRINTTAB(8)"[F7]    STOP RHYTHM"
1440 PRINTTAB(8)"[CLR]   CLEAR CHART"
1450 PRINTTAB(8)"[8T]     END PROGRAM"
1460 PRINTTAB(8)"[<]     MOVE CURSOR LEFT"
1470 PRINTTAB(8)"[>]     MOVE CURSOR RIGHT"
1480 PRINTTAB(8)"[K]     MOVE CURSOR UP"
1490 PRINTTAB(8)"[M]     MOVE CURSOR DOWN"
1500 RETURN
1510 REM *SET UP M/CODE & SPRITE*
1520 FORI=49152TO49413
1530 READJ:C=C+J:POKEI,J:NEXTI
1540 READJ:IFC<>JTHENPRINT"DATA ERROR":END
1550 FORI=0TO62:READJ:POKE832+I,J:NEXTI
1560 VIC=53248:X=28:Y=95:B=1024:SID=54272
1570 FORI=0TO24:POKESID+I,0:NEXTI
1580 POKESID+24,15:POKEVIC+21,1:POKE2040,13
1590 POKE254,15:POKE679,15:POKEVIC+39,1
1600 FORI=0TO50:POKE50000+I,0:NEXT
1610 RETURN
1620 REM***M/C DATA***
1630 DATA120,173,20,3,133,251,173,21,3
1640 DATA133,252,169,36,141,20,3,169
1650 DATA192,141,21,3,88,96,120,165,251
1660 DATA141,20,3,165,252,141,21,3,88
1670 DATA96,32,177,192,165,197,201,3
1680 DATA208,3,32,135,192,165,197,201,4
1690 DATA208,3,32,140,192,165,197,201,5
1700 DATA208,3,32,155,192,165,197,201,6
1710 DATA208,3,32,166,192,32,87,192,142
1720 DATA168,2,140,169,2,76,49,234,165
1730 DATA253,208,1,96,198,254,240,1,96
1740 DATA32,149,192,224,16,208,3,32,144
1750 DATA192,185,80,195,240,3,32,184
1760 DATA192,185,96,195,240,3,32,210
1770 DATA192,185,112,195,240,3,32,236
1780 DATA192,200,234,232,96,169,0,133
1790 DATA253,96,169,1,133,253,162,0,160
1800 DATA0,96,173,167,2,133,254,96,173
1810 DATA167,2,201,255,240,3,238,167,2
1820 DATA96,173,167,2,201,1,240,3,206
1830 DATA167,2,96,174,168,2,172,169,2
1840 DATA96,169,14,141,6,212,169,32,141
1850 DATA2,212,169,66,141,4,212,169,3
1860 DATA141,1,212,169,65,141,4,212,96
1870 DATA169,7,141,12,212,169,12,141,13
1880 DATA212,169,128,141,11,212,169,65
1890 DATA141,8,212,169,129,141,11,212
1900 DATA96,169,2,141,19,212,169,13,141
1910 DATA20,212,169,18,141,18,212,169
1920 DATA100,141,15,212,169,17,141,18
1930 DATA212,96
1940 DATA32038:REM*CHECKSUM*
1950 REM**SPRITE CURSOR DATA**
1960 DATA127,254,0,127,254,0,127,254,0
1970 DATA112,14,0,112,14,0,112,14,0,112
1980 DATA14,0,112,14,0,112,14,0,112,14
1990 DATA0,112,14,0,127,254,0,127,254,0
2000 DATA127,254,0,0,0,0,0,0,0,0,0,0,0
2010 DATA0,0,0,0,0,0,0,0,0,0,0
```

Quellentext in Assembler

```
;++++++++++++++++++++++++
;++++++++++++++++++++++++
;++                    ++
;++  CBM DRUM MACHINE ++
;++    SOURCE CODE     ++
;++                    ++
;++++++++++++++++++++++++
;++++++++++++++++++++++++
;
VOL     =  $D418    ;SID VOLUME
ATT1    =  $D405    ;SID ATTACK VOICE 1
SUS1    =  $D406    ;SID SUSTAIN VOICE 1
PULSE   =  $D402    ;SID PULSE RATE VOICE 1
WAVE1   =  $D404    ;SID WAVEFORM VOICE 1
BASS    =  $D401    ;SID HI-BYTE FREQ VOICE 1
ATT2    =  $D40C    ;SID ATTACK VOICE 1
SUS2    =  $D40D    ;SID SUSTAIN VOICE 2
WAVE2   =  $D40B    ;SID WAVEFORM VOICE 2
SNARE   =  $D408    ;SID HI-BYTE FREQ VOICE 2
ATT3    =  $D413    ;SID ATTACK VOICE 3
SUS3    =  $D414    ;SID SUSTAIN VOICE 3
WAVE3   =  $D412    ;SID WAVEFORM VOICE 3
BELL    =  $D40F    ;SID HI-BYTE FREQ VOICE 3
ROW1    =  $C350    ;STORAGE FOR VOICE 1
ROW2    =  $C360    ;STORAGE FOR VOICE 2
ROW3    =  $C370    ;STORAGE FOR VOICE 3
TEMPO   =  $02A7    ;TEMPORARY STORE FOR DELAY
XCOUNT  =  $02A8    ;TEMPORARY STORE FOR X REGISTER
YCOUNT  =  $02A9    ;TEMPORARY STORE FOR Y REGISTER
LOVEC   =  $FB      ;STORE FOR LO-BYTE VECTOR
HIVEC   =  $FC      ;STORE FOR HI-BYTE VECTOR
PLAY    =  $FD      ;TOGGLE FOR PLAY DRUM (1=YES)
DELAY   =  $FE      ;STORE FOR CURRENT DELAY STATUS
KEY     =  $C5      ;CURRENT KEY PRESSED
;-------------------------------------------------
*       =  $C000    ;ASSEMBLE FROM 49152 (DECIMAL)
;-------------------------------------------------
;SET UP WEDGE
;-------------------------------------------------
        SEI         ;DISABLE INTERRUPT REQUEST
        LDA $0314   ;GET CONTENTS OF VECTOR LO-BYTE
        STA LOVEC   ;STORE IN LOVEC
        LDA $0315   ;GET CONTENTS OF VECTOR HI-BYTE
        STA HIVEC   ;STORE IN HIVEC
        LDA #<WEDGE ;GET LO-BYTE WEDGE START ADDR.
        STA $0314   ;STORE IN IRQ VECTOR LO-BYTE
        LDA #>WEDGE ;GET HI-BYTE OF WEDGE START
        STA $0315   ;STORE IN IRQ VECTOR HI-BYTE
        CLI         ;RESUME INTERRUPT REQUEST
        RTS         ;RETURN
;-------------------------------------------------
;REMOVE WEDGE
;-------------------------------------------------
        SEI         ;DISABLE INTERRUPT REQUEST
        LDA LOVEC   ;GET ORIGINAL VALUE FROM LOVEC
        STA $0314   ;STORE IN IRQ VECTOR LO-BYTE
        LDA HIVEC   ;GET ORIGINAL VALUE FROM HIVEC
        STA $0315   ;STORE IN IRQ VECTOR HI-BYTE
        CLI         ;RESUME INTERRUPT REQUEST
        RTS         ;RETURN
;-------------------------------------------------
;MAIN LOOP
;-------------------------------------------------
WEDGE   JSR REG     ;GOSUB REG
        LDA KEY     ;WHICH KEY PRESSED ?
        CMP #$03    ;IS IT FUNCTION KEY #1 ?
        BNE CONT1   ;BRANCH IF NOT TRUE
        JSR FLAG0   ;GOSUB FLAG0
CONT1   LDA KEY     ;WHICH KEY PRESSED ?
        CMP #$04    ;IS IT FUNCTION KEY #7 ?
        BNE CONT2   ;BRANCH IF NOT TRUE
        JSR FLAG1   ;GOSUB FLAG1
CONT2   LDA KEY     ;WHICH KEY PRESSED ?
        CMP #$05    ;IS IT FUNCTION KEY #3 ?
        BNE CONT3   ;BRANCH IF NOT TRUE
        JSR ADD     ;GOSUB ADD
CONT3   LDA KEY     ;WHICH KEY PRESSED ?
        CMP #$06    ;IS IT FUNCTION KEY #5 ?
        BNE CONT4   ;BRANCH IF NOT TRUE
        JSR MINUS   ;GOSUB MINUS
CONT4   JSR REST    ;GOSUB REST
        STX XCOUNT  ;STORE VALUE OF X REGISTER
        STY YCOUNT  ;STORE VALUE OF Y REGISTER
        JMP $EA31   ;JUMP BACK TO INTERRUPT
;-------------------------------------------------
;CHECK ROUTINE
;-------------------------------------------------
REST    LDA PLAY    ;GET TOGGLE VALUE
        BNE BEGIN   ;IF EQUAL TO 1 THEN BRANCH
        RTS         ;RETURN
;-------------------------------------------------
BEGIN   DEC DELAY   ;DECREMENT DELAY
        BEQ START   ;BRANCH IF ZERO
        RTS         ;RETURN
;-------------------------------------------------
START   JSR COUNT   ;GOSUB COUNT
        CPX #$10    ;END OF LOOP ?
        BNE CHECK   ;BRANCH IF NOT 0
        JSR RESET   ;GOSUB RESET
;-------------------------------------------------
CHECK   LDA ROW1,Y  ;GET VALUE OF ROW1 OFFSET BY Y
        BEQ NEXT1   ;BRANCH IF ZERO
        JSR DRUM1   ;GOSUB DRUM1
NEXT1   LDA ROW2,Y  ;GET VALUE OF ROW2 OFFSET BY Y
        BEQ NEXT2   ;BRANCH IF ZERO
        JSR DRUM2   ;GOSUB DRUM2
NEXT2   LDA ROW3,Y  ;GET VALUE OF ROW3 OFFSET BY Y
        BEQ NEXT3   ;BRANCH IF ZERO
        JSR DRUM3   ;GOSUB DRUM3
NEXT3   INY         ;INCREMENT OFFSET
        INX         ;INCREMENT LOOP COUNTER
        RTS         ;RETURN
;-------------------------------------------------
;SUBROUTINES
;-------------------------------------------------
FLAG0   LDA #$00    ;STORE ZERO -
        STA PLAY    ;IN PLAY
        RTS         ;RETURN
;-------------------------------------------------
FLAG1   LDA #$01    ;STORE 1 -
        STA PLAY    ;IN PLAY
RESET   LDX #$00    ;RESET X REGISTER
        LDY #$00    ;RESET Y REGISTER
        RTS         ;RETURN
;-------------------------------------------------
COUNT   LDA TEMPO   ;GET VALUE OF TEMPO
        STA DELAY   ;STORE IN DELAY
        RTS         ;RETURN
;-------------------------------------------------
ADD     LDA TEMPO   ;GET VALUE OF TEMPO
        CMP #$FF    ;COMPARE RESULT WITH 255
        BEQ CONT5   ;BRANCH IF TRUE
        INC TEMPO   ;INCREMENT TEMPO
CONT5   RTS         ;RETURN
;-------------------------------------------------
MINUS   LDA TEMPO   ;GET VALUE OF TEMPO
        CMP #$01    ;COMPARE RESULT WITH 1
        BEQ CONT6   ;BRANCH IF TRUE
        DEC TEMPO   ;DECREMENT TEMPO
CONT6   RTS         ;RETURN
;-------------------------------------------------
REG     LDX XCOUNT  ;STORE VALUE OF XCOUNT IN X REG
        LDY YCOUNT  ;STORE VALUE OF YCOUNT IN Y REG
        RTS         ;RETURN
;-------------------------------------------------
;DRUM SOUND ROUTINES
;-------------------------------------------------
DRUM1   LDA #$0E    ;SET
        STA SUS1    ;
        LDA #$20    ;UP
        STA PULSE   ;
        LDA #$42    ;BASS
        STA WAVE1   ;
        LDA #$03    ;DRUM
        STA BASS    ;
        LDA #$41    ;AND PLAY
        STA WAVE1   ;
        RTS         ;RETURN
;-------------------------------------------------
DRUM2   LDA #$07    ;SET
        STA ATT2    ;
        LDA #$0C    ;UP
        STA SUS2    ;
        LDA #$80    ;SNARE
        STA WAVE2   ;
        LDA #$41    ;DRUM
        STA SNARE   ;
        LDA #$81    ;AND PLAY
        STA WAVE2   ;
        RTS         ;RETURN
;-------------------------------------------------
DRUM3   LDA #$02    ;SET
        STA ATT3    ;
        LDA #$0D    ;UP
        STA SUS3    ;
        LDA #$12    ;COW
        STA WAVE3   ;
        LDA #$64    ;BELL
        STA BELL    ;
        LDA #$11    ;AND PLAY
        STA WAVE3   ;
        RTS         ;RETURN
        .END
```

Hintertürchen

Über das ROM des Interface 1 können Sie den Sinclair Spectrum mit eigenen BASIC-Befehlen versehen. In der vorigen Folge hatten wir uns die „Mechanik" dieser neuen Befehle angesehen, diesmal zeigen wir, wie sie angelegt werden.

Die neuen BASIC-Befehle müssen bei den Syntax- und Runtimeprüfungen des Haupt-ROMs und des Interface-1-ROMs zunächst einmal Fehlermeldungen erzeugen, da sonst die Steuerung nicht an die von der Variablen VECTOR (bei 23735 und 23736) angegebene Routine übergeben wird. Die Fehler werden mit zwei Techniken hervorgerufen:
1. Sie verändern ein Schlüsselwort, das die Tests dann nicht mehr bestehen kann. Dabei genügt die Eingabe einer falschen Parameterzahl, einer veränderten Syntax oder eines zusätzlichen Zeichens. Wir geben Ihnen dafür Beispiele:
BORDER* (ein an den Befehl angehängtes Zeichen) LINE 10,10,1,19 (zu viele Parameter)
2. Wenn Sie ein Zeichen vor den Befehl stellen, erzeugen Sie einen neuen Befehl, beispielswiese *BEEP oder *PRINT.

Da Befehle dieser Art beliebig auf BASIC-Variablen zugreifen können, erstellen wir unsere neuen Befehle mit der zweiten Methode.

Um etwa den Befehl *B zu definieren, der einen Ton von der Dauer einer Sekunde erzeugen soll, muß zunächst entschieden werden, wo der neue Code untergebracht werden soll. Ein REM-Befehl kommt nicht in Frage, da die Systemvariablen des Interface 1 nicht immer mit den gleichen Speicheradressen arbeiten. Mit dem Befehl CLEAR nn schaffen Sie jedoch den notwendigen Platz.

Auf Fehler reagiert der Spectrum mit RST #08. Sobald die Hardware des Interface 1 feststellt, daß der Programmzähler des Z80 diese Adresse enthält, wird das ROM des Interface 1 aktiviert und ein Sprung auf die von VECTOR angezeigte Routine ausgeführt.

Systembaukasten

Jeder neue Befehl muß bestimmte „Bauteile" enthalten (siehe Bild). Wenn nun der BASIC-Interpreter einen Fehler erkennt und die von VECTOR angegebene Routine aufruft, sollte man zuerst untersuchen, welches Zeichen des BASIC-Programms das Problem verursacht hat.

Praktischerweise zeigt die Systemvariable CHADD (bei 23755 und 23756) auf dieses Zeichen. Mit zwei Routinen des Haupt-ROMs – GETCHAR und NEXTCHAR (siehe Kasten „Nützliche Adressen") – holen Sie sich das Zeichen. Die Routinen werden aus dem Bereich des Schatten-ROMs mit RST #10 gefolgt von #0018 (GETCHAR) oder #0020 (NEXTCHAR) angesprochen.

Beim Aufruf der Routine, auf die VECTOR zeigt, enthält CHADD die Adresse des Zeichens, das den Fehler ausgelöst hat. Der Befehl *B besteht aus folgendem Code:

```
D7      vector: rst  #10
1800            defw #0018              ;address of GETCHAR
FE2A            cp   "*"                ;is it a "*"?
C2F001          jp   nz,#01F0           ;if not, error
D7              rst  #10
2000            defw #0020              ;address of NEXTCHAR
FE42            cp   "B"                ;is it a "B"
C2F001          jp   nz,#01F0           ;if not, error
D7              rst  #10                ;get NEXTCHAR, should...
2000            defw #0020              ; ...be end of statement
CDB705          call #05B7              ;check end of statement
        runtime:
```

Vor Aufruf der Schatten-ROM-Routine bei #05B7 muß das Zeichen, auf das CHADD zeigt, im A-Register gespeichert werden. Das gleiche Ergebnis läßt sich auch mit NEXTCHAR erreichen, doch veranlaßt die Syntaxprüfung in der Routine bei #05B7 einen Rücksprung zum Haupt-ROM, um dort die nächste Anweisung zu testen. Während des Ablaufs (Runtime) steht hier jedoch ein normaler Subroutinenrücksprung, der den Code beim Label RUNTIME anspricht.

Beachten Sie, daß die Befehle Zeichen für Zeichen geprüft werden. Sie können die Routine leicht umbauen und so die Befehlseingabe mit Groß- und Kleinbuchstaben möglich machen. Wenn ein Zeichen den Test nicht besteht, wird wie bei einer Fehlermeldung das Schatten-ROM über #01F0 (Version 1) angesprochen. Sie müssen das aktuelle von CHADD angegebene Zeichen hier allerdings mit GETCHAR statt mit NEXTCHAR holen.

Der neue Befehl erzeugt den Ton über die BEEP-Routine des Haupt-ROMs:

```
11050I runtim: ld   de,#0105           ;duration of 1 second
210007         ld   hl,#0700           ;pitch
D7             rst  #10
B503           defw #03B5              ;execute BEEP routine
C3C105         jp   #05C1              ;exit
```

Eine Verzweigung zu #05C1 gibt die Steuerung an das Haupt-ROM zurück.

Ihre neuen BASIC-Befehle können – über RST #10 – natürlich auch mit anderen Routinen des Haupt-ROMs und den Routinen des Grafik-ROMs (wir gehen später genauer darauf ein) arbeiten. So setzt das folgende Listing des *B-Befehls die Routinen bei #1C82 und #1E94 (siehe Kasten) zur Bewertung eines numerischen Parameters ein, der vom Anwender angegeben wurde. Das Befehlsformat lautet *B n,

wobei n zwischen 0 und 255 liegen kann. Der Parameter gibt die Länge des BEEP in Sekunden an; *B 255 erzeugt einen Ton von 255 Sekunden.

```
                org  60000
       VECTOR:  equ  #5CB7              ;VECTOR address
CF     init:    rst  #8
31              defb 49                 ;set up IF1 variables
2169EA          ld   hl,newcom          ;add of new command
22B75C          ld   (VECTOR),hl        ;get it in VECTOR
C9              ret                     ;end of initialisation

D7     newcom:  rst  #10                ;VECTOR jumps here
1800            defw #0018              ;GETCHAR
FE2A            cp   "*"
C2F001          jp   nz,#01F0
D7              rst  #10
2000            defw #0020              ;NEXTCHAR
FE42            cp   "B"
C2F001          jp   nz,#01F0
D7              rst  #10
2000            defw #0020
D7              rst  #10
821C            defw #1C82              ;evaluate parameter
CDB705          call #05B7              ;check statement end
D7     runtim:  rst  #10
941E            defw #1E94              ;get param in a
47              ld   b,a
C5     loop:    push bc                 ;save counter
11B501          ld   de,#0185           ;duration
210007          ld   hl,#0700           ;pitch
D7              rst  #10
B503            defw #03B5              ;call BEEP routine
C1              pop  bc                 ;restore counter
10F3            djnz loop
C3C105          jp   #05C1              ;exit - all done
```

Da die Routine bei #1C82 nur Acht-Bit-Werte akzeptiert (sie hat für diesen Wert nur die acht Bits des A-Registers zur Verfügung), erzeugen Zahlen außerhalb des Bereichs von 0 bis 255 einen Fehler. Denken sie auch daran, daß der Zähler FRAMES während der Tonerzeugung nicht inkrementiert wird und damit auch die Tastaturabfrage abgeschaltet ist. Bei Angabe des Parameters 255 sollten Sie sich daher auf einige Wartezeit einrichten.

Bisher wurden die Fehler durch Eingaben erzeugt, die nicht dem Standard entsprachen. Sie können aber auch die normalen Schlüsselworte des Sinclair-BASIC so modifizieren, daß dadurch Fehler hervorgerufen werden. So rufen die Zusatzparameter der folgenden Beispiele einen Sprung auf die von VECTOR angegebene Routine hervor:

CIRCLE x,y,z,n
LINE x
BORDER x,y,z

Hier zeigt CHADD beim Einsprung in die VECTOR-Routine auf das „fehlerhafte" Zeichen, das mit dem entsprechenden Tokencode verglichen wurde.

Trotz der recht komplizierten Technik lassen sich in das Sinclair-BASIC leicht neue Befehle einbauen. Sie müssen nur genau wissen, welches ROM zur Zeit aktiv ist. Das Schatten-ROM läßt sich vom Haupt-ROM aus jederzeit mit dem Hakencode 50 (und der gewünschten Adresse) ansprechen.

Interface-1-Versionen

Wenn Sie direkt auf Routinen des Schatten-ROMs zugreifen wollen, sollten sie zuvor feststellen, mit welcher Version des Interface 1 Sie arbeiten. Generell sind alle IF1 mit Seriennummern über 87315 mit der ROM-Version 2 ausgestattet. Die Eingabe des folgenden Befehl ist jedoch zuverlässiger:

CLOSE #0;PRINT PEEK 23729

Bei einem Ergebnis 0 liegt Version 1 vor, bei 80 die Version 2. Glücklicherweise haben alle ROM-Routinen, die wir zur Erstellung neuer Befehle angesprochen haben, in beiden Versionen die gleichen Adressen. Andere Routinen dürften jedoch Schwierigkeiten machen. Mit einem geeigneten Disassembler und unserem Ladeprogramm für das Schatten-ROM (siehe letzte Folge) können Sie sich den Inhalt der Routinen anzeigen lassen.

Das ROM der Version 2 enthält zwei neue Hakencodes. Hakencode #33 lädt die Datensatzbeschreibung einer Datei und Hakencode #34 eröffnet einen RS 232 „b" Kanal.

Nützliche Adressen

Im Schatten-ROM:

#01F0 Verläßt das Schatten-ROM über die Fehlerroutine des Haupt-ROMs

Verläßt das Schatten-ROM, wenn das A-Register #0D (ENTER) oder ein Semikolon enthält. Sonst: Fehler.

#05C1 Kehrt bei neuer Befehlsannahme zum Interpreter zurück.

Im Haupt-ROM:

#0018 GETCHAR – Holt CHADD (siehe unten) aus HL und speichert das Zeichen, auf das CHADD zeigt, in A. Prüft, ob das Zeichen angezeigt werden kann und veranlaßt gegebenenfalls einen Rücksprung.

#0020 NEXTCHAR – Inkrementiert CHADD und speichert den Wert in HL. Holt das Zeichen, auf das CHADD zeigt, und prüft, ob es angezeigt werden kann. Ist dies der Fall, wird das Zeichen in A gespeichert und ein Rücksprung veranlaßt.

#1C82 Bewertet oder prüft einen numerischen Ausdruck, auf dessen erstes Element CHADD zeigt. Beim Rücksprung zeigt CHADD auf das erste Zeichen hinter dem Ausdruck, während das Ergebnis sich auf dem Berechnungsstack des Haupt-ROMs befindet.

#1E94 Nimmt einen Wert vom Berechnungsstack des Haupt-ROMs und stellt ihn in A, wenn er zwischen 0 und 255 liegt.
Sonst: Fehler.
Die Systemvariable CHADD (bei 23755 und 23756) zeigt auf das interpretierte Zeichen.

Perfekt im Bild

Das Betriebssystem des Spectrum bietet dem Maschinencodeprogrammierer interessante Möglichkeiten zur Steuerung des Bildschirms. Wir untersuchen die Dateien für Bildschirmdarstellung und fassen die Systemroutinen zusammen.

Die Grafik- und Bildschirmsteuerung des Spectrum ist nicht besonders kompliziert. Wir sehen uns zunächst die ROM-Routinen an, die das OS des Spectrum dafür bietet.

Um den Bildschirm adressieren zu können, muß als erstes Kanal S durch die Anwahl von Strom 2 eröffnet werden. Für einen einfachen Anzeigevorgang ist RST#10 bei weitem die einfachste Methode. Für die Darstellung eines ganzen Zeichenstrings müssen Sie jedoch die Zeichen mit einer kurzen Routine einlesen und einzeln an die RST#10-Routine übergeben.

Die entsprechende Routine ist im ROM abgelegt. Für den Aufruf laden Sie DE mit der Adresse des Strings, BC mit der Stringlänge und rufen dann die „Print String"-Routine bei #203C auf. Unser erstes Listing zeigt den Ablauf in der Praxis:

```
                ORG  60000
1172EA          LD   DE,MSG
3E02            LD   A,2              ;select stream 2
CD0116          CALL 0x1601           ;open channel S
1172EA          LD   DE,MSG           ;point DE to string
012600          LD   BC,38            ;load BC with string len
CD3C20          CALL 0x203C           ;call PRINT-STRING
C9              RET
160A06   MSG:   DEFB 22,10,6          ;codes for AT 10,5
54484520        DEFM "THE HOME COMPUTER"
160C07          DEFB 22,12,7
41445641        DEFM "ADVANCED COURSE"
```

Praktisch ist auch die ROM-Routine bei #2DE3, die eine Zahl im Fließkommaformat anzeigt. Wir haben bereits gesehen, daß die Routine bei #1C82 einen numerischen Ausdruck bewertet und das Ergebnis auf den „Calculator Stack" schiebt. Ohne diese nützliche Routine ist die Darstellung einer Zahl vom Maschinencode aus recht kompliziert. Die Zahl wird zunächst in das Registerpaar BC oder in das A-Register geladen und dann entweder mit #2D28 die STACK-A-Routine oder mit #2D2B die STACK-BC-Routine aufgerufen, die die Zahl auf den Stack schiebt. Danach brauchen Sie nur noch die Routine der Fließkommaanzeige aufzurufen.

```
                ORG  60000
013930          LD   BC,12345         ;get number in BC
CD2B2D          CALL 0x2D2B           ;BC onto calc stack
3E02            LD   A,2
CD0116          CALL 0x1601           ;open channel S
CDE32D          CALL 0x2DE3           ;print no from stack top
C9              RET
```

Die Vielseitigkeit der hochauflösenden Grafik des Spectrum zeigt sich jedoch erst beim Einsatz der Befehle PLOT und DRAW. PLOT hatten wir früher schon behandelt. DRAW ist ebenso einfach, benötigt aber mehr Parameter. Der Befehl zeichnet die kürzeste Linie zwischen dem zuletzt gezeichneten Punkt und einem Punkt, der relativ zum Ursprung definiert wurde. Für das Zeichen einer Linie 20 Pixel nach oben und 40 Pixel nach links lautet der Befehl ganz einfach DRAW −40,20.

Vom Maschinencode aus hat der DRAW-Befehl das Format DRAW x,y, wobei x und y separat behandelt werden müssen. Zur Vereinfachung lassen wir Kurvenfaktoren erst einmal beiseite. Wenn eine Rückkehr zum BASIC geplant ist, muß der Wert des alternativen Registerpaares H'L' erhalten bleiben, da die DRAW-Routine diese Daten verändert.

Der absolute Wert von x (ABS x) wird in das C-Register gestellt, ABS y in das B-Register und die Vorzeichenwerte von x und y entsprechend in D und E. Die Werte dürfen über −1, 0 und 1 liegen, da die Routine sie als Inkrement zur letzten geplotteten Pixelposition addiert, um die nächste Position zu berechnen.

Zwei Linientypen

Bei einem Vorzeichenwert von 2 oder 3 ist die Linie dann gepunktet, aber auch zwei- bzw. dreimal länger als bei Inkrement 1. Beachten Sie, daß die Vorzeichenwerte das Format des Zweierkomplements haben müssen. Unser drittes Beispiel zeigt, wie die beiden Linientypen gezeichnet werden.

```
                ORG  60000
D9              EXX
E5              PUSH HL               ;save H'L'
D9              EXX
0658            LD   B,88             ;Y coordinate
0E64            LD   C,100            ;X coordinate
CDE522          CALL 0x22E5           ;PLOT 100,88
0E28            LD   C,40             ;ABS x=40
0614            LD   B,20             ;ABS y=20
1EFF            LD   E,255            ;SGN y=-1
1601            LD   D,1              ;SGN x=1
CDBA24          CALL 0x24BA           ;DRAW -40,20
3E58            LD   A,88
327D5C          LD   (23677),A        ;X org=88
3E32            LD   A,50
327E5C          LD   (23678),A        ;Y org=50
0E28            LD   C,40
0614            LD   B,20
1602            LD   D,2              ;SGN x=2 - stipple line
1E02            LD   E,2              ;SGN y=2 - stipple line
CDBA24          CALL 0x24BA
D9              EXX
E1              POP  HL               ;restore H'L'
D9              EXX
C9              RET
```

Unsere beiden Tabellen auf Seite 2128 enthalten in Kurzform, was wir eben über Grafik und Bildschirmanzeige gesagt haben. Eine Tabelle zeigt die Routinen und die andere praktische Systemvariablen für den Einsatz der Routinen.

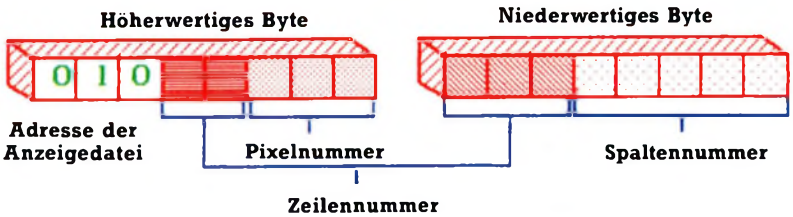
Abteilungsleiter

Der Bildschirmspeicher des Spectrum besteht aus zwei Abteilungen. Der erste Teil heißt „Anzeigedatei" und enthält Daten über jedes einzelne Pixel. Der zweite Teil ist die „Attributdatei" mit der Farbinformation. Die Anzeigedatei ist 6144 Bytes lang und erstreckt sich von 16384 bis 22527. Geben Sie das folgende Programm ein, und rufen Sie es auf:

```
10 FOR F=16384 TO 22527
20 POKE F,255
30 NEXT F
```

Der Bildschirm füllt sich langsam in drei Abschnitten. Wenn Sie nahe herangehen, können Sie erkennen, daß die Linien pixelweise gezeichnet werden, aber nicht aufeinander folgen: Die erste Pixelzeile erscheint am oberen Bildschirmrand, die nächste acht Linien darunter, die dritte sechzehn Linien unter der ersten etc. Nach der vierundsechzigsten Linie beginnt der Vorgang wieder von oben und füllt die zweite Linie, danach die neunte usw. Dieser Ablauf wiederholt sich auch im zweiten und dritten Drittel des Schirms (siehe Diagramm auf dieser Seite ganz unten).

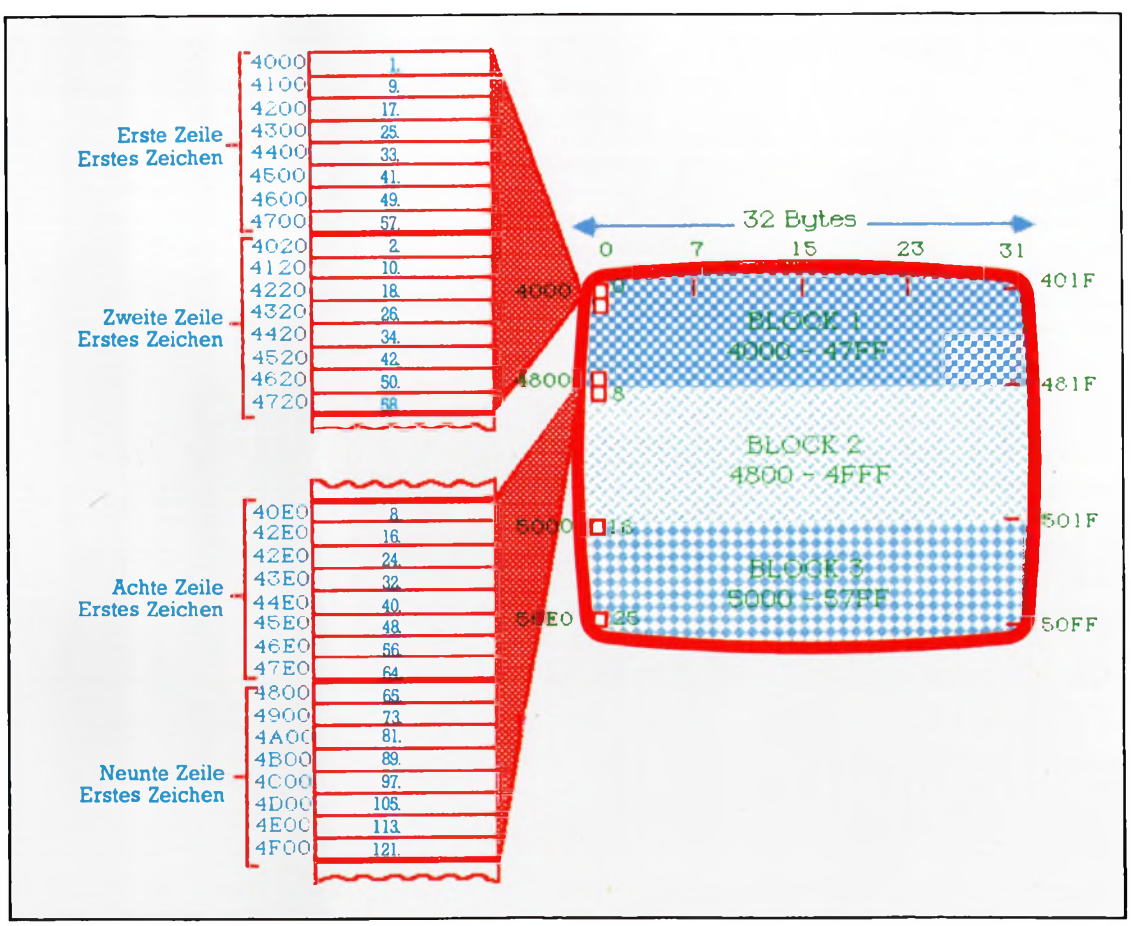
Für den Einsatz der Anzeige- bzw. Displaydatei (auch „DFile" genannt) müssen Sie berechnen, welche Adressen welche Pixelpositionen ansprechen. Das Bild zeigt, wie die Bits einer DFile-Adresse angeordnet sind. Dabei bezieht sich die Pixelnummer auf die Pixelzeile innerhalb eines Zeichenfeldes. Das oberste Pixel eines Zeichens hat die Pixelnummer 0.



Die Struktur der einzelnen Bits einer DFile-Adresse sieht zwar kompliziert aus, doch lassen sich die Bildschirmdaten damit schnell durch einzelne Registerbefehle verändern. Die Verarbeitung mit Registerpaaren wäre weit umständlicher und langsamer.

Auf die Schnelle

Wenn Sie sich in einem Zeichenfeld von einer Pixelzeile auf die darunterliegende Zeile bewegen möchten, brauchen Sie nur das Register H zu inkrementieren (INC H). Für die Bewegung um ein Zeichen nach rechts wird INC HL eingesetzt. Die Position einer PRINT-AT-Koordinate läßt sich mit der nächsten Routine berechnen. Dafür wird die Spaltennummer in E und die Zeilennummer in D gespeichert. Nach Ablauf der Routine steht die gewünschte Adresse in HL.



So ist die Anzeigedatei des Spectrum aufgebaut. Der Bildschirm unterteilt sich in drei Blöcke, die je 256 Zeichenfelder (acht Zeilen zu je 32 Zeichen) darstellen. In jedem Block sind die Pixelzeilen wie im Bild gezeigt untergebracht: Der erste Block von 256 Bytes enthält die Daten der ersten Pixelzeile der ersten 256 Zeichenfelder, die zweiten 256 Bytes die Daten der zweiten Pixelzeile etc. Im vergrößerten Bildausschnitt sind die vier Zeichen auf der linken Bildschirmseite zu erkennen. Die Pixelzeilen jedes Zeichenfeldes sind durch die entsprechenden Adressen der Anzeigedatei festgelegt.

Spectrum Variable

Name	Adresse	Beschreibung
COORDS	23677	Enthält X-Koordinate des zuletzt geplotteten Punktes
COORDS	23678	Enthält Y-Koordinate des zuletzt geplotteten Punktes
DF CC	23684	Eine Zwei-Byte-Variable mit der Speicheradresse des ersten Anzeigedateibytes, das angezeigt werden soll.
S POSN	23688	Die aktuelle Anzeigeposition für Spalten
S POSN	23689	Die aktuelle Anzeigeposition für Zeilen
ATTR P	23693	Speichert die permanenten Farben, die von INK, PAPER, FLASH und BRIGHT gesetzt wurden. Die Daten haben das ATTR-Format
MASK P	23694	Maskiert ATTR P, wenn transparente Farben (INK 9 etc.) eingesetzt werden. Für jedes gesetzte Bit wird das entsprechende Bit von ATTR P ignoriert.
ATTR T	23695	Wie ATTR P, jedoch nur für temporäre Farben – z. B. bei Farbänderungen innerhalb eines PRINT-Befehls
MASK T	23696	Wie MASK P, jedoch für temporäre Farben
P FLAG	23697	Verschiedene Flags für den Status der temporären und permanenten Farben. Jedes Bit zeigt, ob INK oder PAPER temporär oder permanent ist.

ATTR P, MASK P und P FLAG haben wir früher schon ausführlich beschrieben.

Spectrum ROM Routinen

Name	Adresse	Einsprungsbedingungen	Ergebnis
RST # 10	#0010	In A muß der ASCII-Code des anzuzeigenden Zeichens stehen	Das Zeichen wird angezeigt
PR STRING	#203C	IN BC muß die Stringlänge stehen und in DF die Position des Strings im Speicher	Der String wird auf der nächsten verfügbaren Schirmposition angezeigt
PRINTED FP	#2DE3	Die anzuzeigende Zahl muß oben auf dem Calculatorstack liegen	Die oberste Zahl des Calculatorstacks wird angezeigt
STK-A	#2D28	A muß den Wert enthalten, der auf den Stack geschoben werden soll	Der Wert von A wird oben auf den Calculatorstack geschoben
STK-BC	#2D2B	BC muß den Wert enthalten, der auf den Stack geschoben werden soll	Der Wert von BC wird auf Calculator geschoben
PLOT	#22E5	In B und C müssen die Y- und X-Koordinaten gespeichert sein	Der Punkt (x,y) wird geplottet und SV COORDS aktualisiert
DRAW	#24BA	B muß ABS (y) enthalten C muß ABS (x) enthalten D muß SGN (y) enthalten E muß SGN (x) enthalten	Wenn die Werte über 1 liegen, wird eine gepunktete Linie gezeichnet
CHAN-OPEN	#1601	A muß die Nummer des zu eröffnenden Streams enthalten	Eröffnet den angegebenen Kanal

```
                ORG  60000
3EAF            LD   A,175
92              SUB  D
57              LD   D,A          ;D holds 175 - y coord
A7              AND  A
1F              RRA
37              SCF
1F              RRA
A7              AND  A
1F              RRA               ;shuffle bits along
AA              XOR  D
E6F8            AND  248
AA              XOR  D
67              LD   H,A          ;H becomes...
78              LD   A,E          ;64*8*INT(B/64)+B(MOD 8)
CB07            RLC  A
CB07            RLC  A
CB07            RLC  A
AA              XOR  D
CB07            RLC  A
CB07            RLC  A            ;shuffle bits along
6F              LD   L,A          ;L now has 32*INT(...
78              LD   A,E          ;...B(MOD64)/8)+INT(x/8)
E607            AND  7            ;lo-byte is x(MOD 8)
C9              RET
```

Pixeladressen können Sie jedoch nur mit der folgenden Routine berechnen. Dabei wird wie beim vorigen Listing D mit der Y-Koordinate und E mit der X-Koordinate geladen. Nach Ablauf der Routine steht die Adresse in HL. Sie enthält einen Wert zwischen 0 und 7, der dem gesetzten Pixelbit entspricht.

```
                ORG  60000
7A              LD   A,D
E6F8            AND  248          ;mask off bits 0-2
CBF7            SET  6,A
67              LD   H,A
7A              LD   A,D
E607            AND  7            ;mask off bits 3-7
CB0F            RRC  A
CB0F            RRC  A
CB0F            RRC  A            ;move bits up
83              ADD  A,E          ;add on column bits
6F              LD   L,A
C9              RET
```

Denken Sie beim DFile auch daran, daß jedes mit POKE abgerufene Byte sich auf acht Pixel bezieht. Die Pixelbytes haben das gleiche Format wie die der anwenderdefinierten Grafik.

Die Attributdatei – der zweite Teil des Bildschirmspeichers – ist einfacher aufgebaut als die Anzeigedatei. Die Datei ist 768 Bytes lang und fängt bei 22528 an. Mit dem Programm

```
10 FOR F=22528 TO 22528 + 767
20 POKE F,0
30 NEXT F
```

können Sie sehen, daß der Bildschirm sich Zeile für Zeile füllt. Hier wird allerdings nach Abschluß der ersten Zeile gleich das erste Byte der zweiten Zeile angesprochen und Zeile für Zeile fortgefahren.

Jedes Byte der Attributdatei bezieht sich auf die INK-, PAPER-, BRIGHT- und FLASH-Werte der einzelnen Zeichenfelder. Die ersten drei Bits enthalten die Farbe von INK, Bit 3, 4 und 5 die von PAPER. Bit 6 gibt Auskunft über BRIGHT und Bit 7 über FLASH.

Zum Schluß noch ein Hinweis: Für eine Änderung der BORDER-Farbe vom Maschinencode aus müssen Sie A mit der Farbnummer laden und #229B aufrufen.

Schneiders Maschinencode

Am Anfang dieser neuen Serie über das Betriebssystem der Schneider CPC 464 und 664 untersuchen wir zunächst, wie RAM und ROM ein- und ausgeblendet werden. Weiterhin verschaffen wir uns über einen speziellen Speicherbereich – den „Jumpblock" – Zugang zu den Firmware-Routinen.

Die Schneider-Computer CPC 464 und 664 enthalten zwar keine außergewöhnlichen technischen Neuerungen, doch gleicht ein umfassend angelegtes Betriebssystem diesen Mangel aus, so daß die Geräte mit minimaler Hardware auskommen. Das Ergebnis ist ein preisgünstiger Micro mit hoher Leistung.

Der CPC 464 arbeitet mit einer Z80A CPU, 64 KByte RAM, 32 KByte ROM, einem 8255 PIA (Peripheral Interface Adaptor), einem AY-3-8912 Klanggenerator, einer 6845 CRTC (Bildschirmsteuerung) und einer speziellen Logikschaltung (ULA – Uncommitted Logic Array).

Das Betriebssystem, auch „Firmware" genannt, belegt die untere Hälfte des ROM – in der oberen Hälfte befindet sich die Schneider-Version von BASIC (das Locomotive BASIC).

Die Memory-Map zeigt, daß der 32 KByte umfassende ROM-Chip zwei logisch getrennte Bereiche zu je 16 KByte enthält, die sich über die ULA unabhängig voneinander ein- und ausschalten lassen. Die beiden Bereiche ersetzen die ersten und letzten 16 KByte des RAM, während die mittleren 32 KByte zum Schreiben und Lesen zur Verfügung stehen. Die ULA kann außerdem in das obere ROM bis zu 252 Zusatz-ROMs einschalten.

Der Bildschirm wird von einem 6845 Chip gesteuert und belegt 16 KByte RAM – normalerweise direkt unter der RAM-Obergrenze. Die Bildschirmdaten lassen sich jedoch in jeden der vier 16K-Blöcke übertragen, so daß mehrere Bildschirminhalte blitzschnell gegeneinander ausgetauscht werden können.

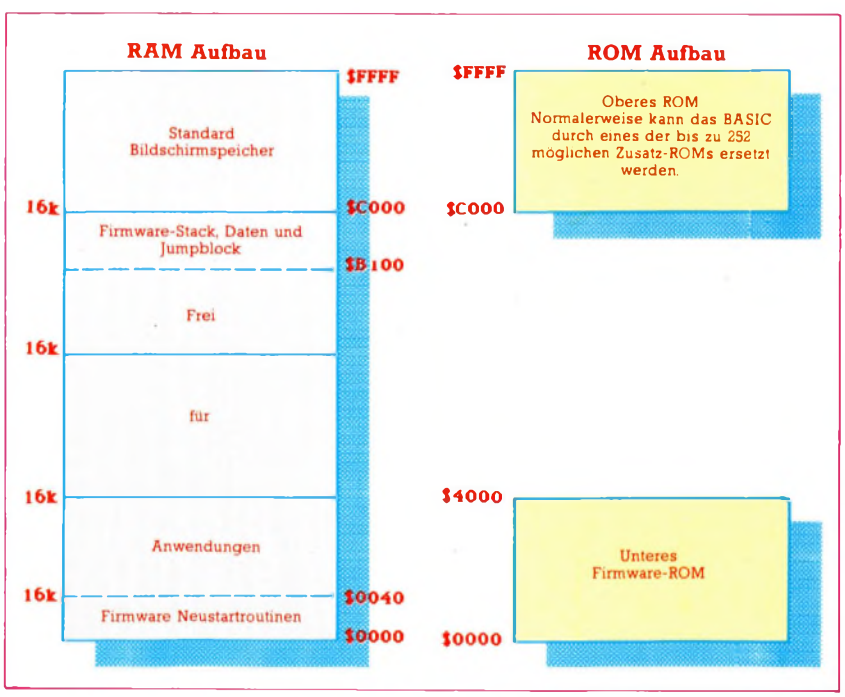
Wenn das obere ROM eingeschaltet ist – etwa bei aktiviertem BASIC – veranlaßt die ULA, daß alle Daten der oberen 16 KBytes aus dem ROM gelesen werden. Alles, was in diesen Block geschrieben werden soll, muß das ROM passieren, um das darunterliegende RAM (normalerweise der Bildschirmspeicher) erreichen zu können. Das obere ROM läßt sich auf mehrere Arten ausschalten, wenn die oberen 16 KBytes RAM gelesen werden sollen.

Auch das untere Firmware-ROM wird auf diese Weise angesprochen. Wie aber kann das Betriebssystem funktionieren, wenn das untere ROM ausgeschaltet ist? Die Lösung ist einfach: Das untere ROM wird von den Programmen nicht direkt angesprochen. Alle Aufrufe an das Betriebssystem gehen über einen „Jumpblock", der in dem festen Speicherbereich liegt. Er enthält Einträge, die einen Sprung (oder einen vergleichbaren Vorgang) auf die Routinen der Firmware auslösen. Durch den Jumpblock lassen sich die Adressen der Firmware-Routinen verändern, ohne daß bestehende Programme in Mitleidenschaft gezogen werden. Sie sollten Firmware-Routinen nur über den Jumpblock aufrufen.

Der Jumpblock liegt im dritten der 16K Blöcke und kann nicht wie ein ROM ein- und ausgeschaltet werden. Normalerweise läßt sich dieser Bereich auch nicht vom Anwender überschreiben. Es gibt jedoch Situationen, in denen der Jumpblock geändert werden muß, damit ein Teil des Betriebssystems verschoben oder ausgetauscht werden kann.

Alle Einsprungspunkte des Betriebssystems sind im Jumpblock eingetragen. Ein Zugriff ruft

Die Memory-Map des Schneider zeigt, wie der 32K-ROM-Chip die oberen und unteren 16K des RAM „ausschaltet". Die Firmware kann beide RAM-Seiten ein- und ausblenden und auch prüfen, ob die bis zu 252 Zusatz-ROMs vorhanden sind. Die Bildschirmdaten lassen sich in jeden der vier 16K-RAM-Blöcke legen.



zuerst den entsprechenden Eintrag des Jumpblocks auf. Dieser gibt einen RST-Befehl aus, der wiederum die Firmware-Routine aktiviert. Die Einträge sind jeweils drei Bytes lang und enthalten beim Einschalten des Systems einen RST-Befehl gefolgt von der Einsprungsadresse des Betriebssystems.

Wenn es nicht benötigt wird, kann das untere ROM ausgeschaltet werden. Im RAM befindet sich eine Kopie der Neustart-Routinen, die das Firmware-ROM vor jedem Zugriff wieder anschalten. Nach Ablauf der Betriebssystemroutinen werden die ROMs in ihren ursprünglichen Zustand versetzt. Damit aber ist die Frage gelöst, welche ROMs vor Aufruf einer Betriebssystemroutine eingeschaltet werden müssen. Die meisten Neustart-Befehle stellen zusätzliche Kommandos dar, die hauptsächlich das Schalten der ROMs steuern.

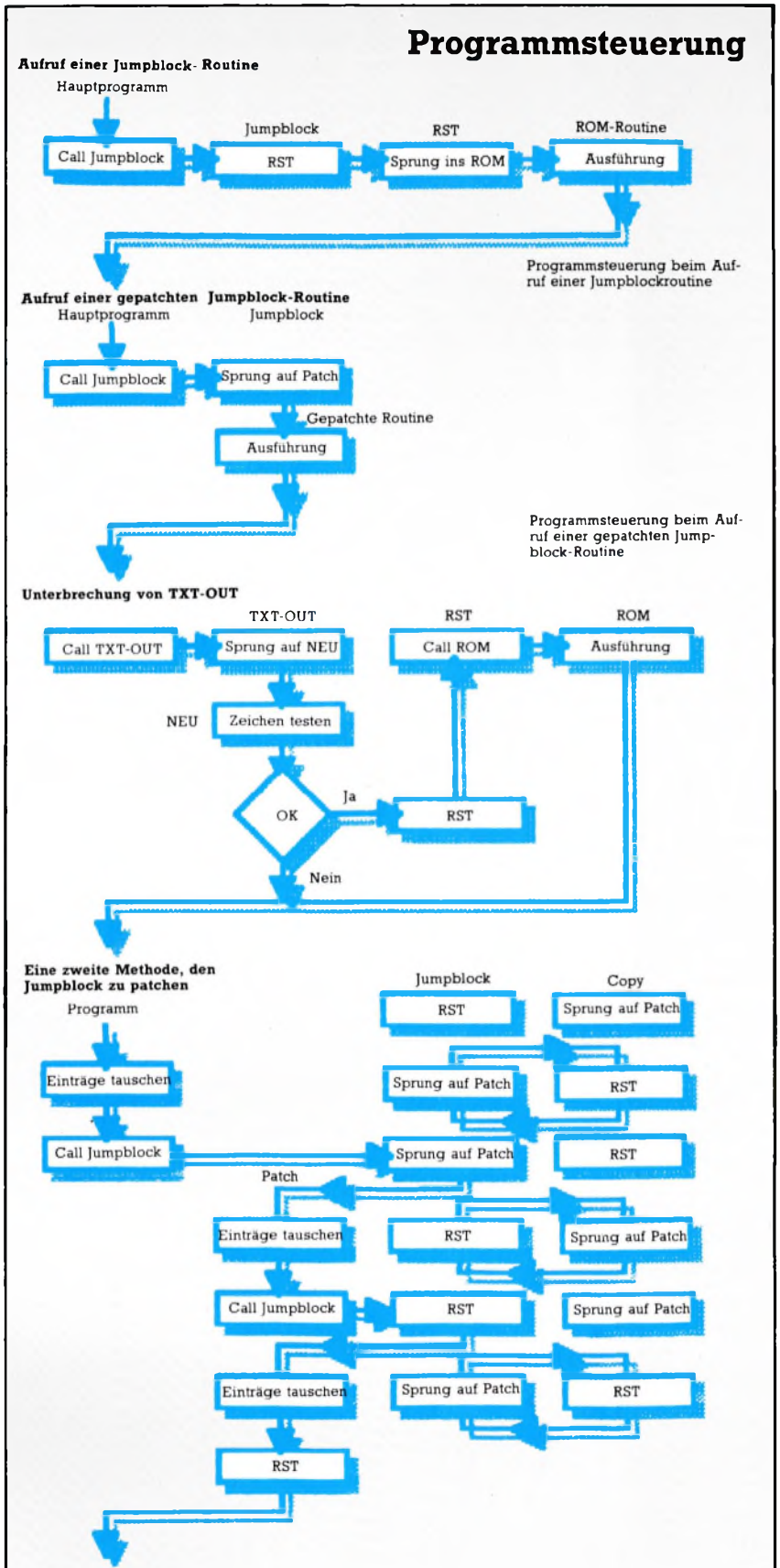
Per Jumpblock lassen sich anstelle der Systemroutinen auch andere Routinen einsetzen. So können Sie beispielsweise eigene Bildschirmmodule einfügen, indem Sie den entsprechenden Jumpblock-Eintrag auf die neue Routine zeigen lassen. Das Bild zur Programmsteuerung zeigt die unterschiedliche Steuerung bei einer normalen Befehlsübergabe an das Betriebssystem und bei einer vom Anwender eingesetzten Unterbrechung. Mit dieser Methode des „Patchens" wird auf dem CPC 464 und 664 auch zwischen den Cassetten- und Diskettenbefehlen umgeschaltet.

An den Bildschirm

Nehmen Sie an, Sie haben eine Routine namens SEND SCREEN geschrieben, die Text zum Bildschirm sendet, und die Sie statt der entsprechenden Betriebssystemroutine einsetzen wollen. Der Jumpblock-Eintrag für die Zeichendarstellung auf dem Bildschirm heißt TXT WR CHAR und wird über OBB5DH adressiert. Patch trägt einfach die Adresse der neuen Routine in den Jumpblock ein.

Die eben beschriebene Routine funktioniert ohne Probleme, wenn eine neue Routine das im Jumpblock eingetragene Original vollständig ersetzt. Schwierigkeiten gibt es, wenn der ursprüngliche Eintrag ebenfalls benötigt wird oder von der neuen Routine aus eine weitere Routine des Jumpblocks aufgerufen wird. Hier ein Beispielprogramm in BASIC:

```
10  MODE 1                    Bildschirm löschen
                              und Cursor nach links
                              oben
20  WHILE -1                  Endlosschleife star-
                              ten
30  INPUT "Zeichen-           Zeichencode holen
    code";C;
40  IF C<0 OR C>              Überprüfung des
    255 then ?:GOTO 30        Zeichens
50  ?CHR$(C);                 Zeichen senden
60  WEND                      Zurück zum Schlei-
                              fenanfang
```



Gehupft wie gesprungen
Über den Jumpblock kann der Anwender problemlos auf die Firmware zugreifen und Systemaufrufe leicht auf seine eigenen Routinen leiten. Dabei muß der Jumpblock „gepatcht" werden. Das Bild zeigt die Abläufe der verschiedenen Methoden.

Mit diesem Programm lassen sich alle Grafikzeichen und auch die Steuercodes anzeigen. Es gibt jedoch einen Code, der den Bildschirm abschaltet und damit das Problem „aufhängt". Sie können das Senden dieses Codes leicht verhindern, wenn Sie in Zeile 40 einen Test einsetzen. In unserem Beispiel werden wir den Test jedoch in Zeile 50 (in der das Zeichen angezeigt wird) durchführen, damit wir das Zeichen über den Jumpblock-Eintrag TXT OUTPUT (bei $BB5A) abfangen können, bevor es zum Bildschirm gesandt wird. Die Originalroutine verlangt, daß das auszugebende Zeichen im Register A gespeichert wird und alle Register vor dem Rücksprung gesichert werden.

Das folgende Programmlisting zeigt die einfachste Methode, einen Jumpblock-Patch einzurichten:

```
        txt_ou: equ  #BB5A              ;jblock output routine
        disabl: equ  21                 ;screen disable char
        ; New routine follows - char to be displayed is given in A
FE15    new:    cp   disable            ;is it disable VDU?
C8              ret  z                  ;if so ignore it
000000  entry:  defb 0,0,0              ;insert original entry
        ;The return address is already on the stack on entry
        ;to the routine, so no RET instruction required
        ;Existing jumpblock entry at txt_out should be copied to ENTRY
        ;Original entry should be replaced with JMP to NEW
```

Sie funktioniert bei allen im Jumpblock eingetragenen Standardroutinen. Bevor Sie eine Routine „patchen", müssen Sie jedoch sicherstellen, daß sie nicht bereits zuvor auf die gleiche Weise bearbeitet wurde. Einige Routinen des Jumpblock funktionieren nur von ihrer Originaladresse aus – wird der Eintrag kopiert, arbeitet sie nicht mehr korrekt.

Unser Bild zeigt auch die alternative Methode, einen Jumpblock zu patchen. Dabei wird der aktuelle Eintrag kopiert und an seine Stelle ein Sprung auf die neue Routine gesetzt. Die neue Routine stellt schließlich den ursprünglichen Eintrag wieder an seinen angestammten Platz und aktiviert ihn. Vor dem Rücksprung ins Programm wird der Jumpblock-Eintrag dann wieder auf die neue Routine gesetzt.

Unser zweites Maschinencodelisting arbeitet mit der zweiten Methode. Sie erscheint auf den ersten Blick zwar umständlicher, funktioniert aber bei allen Jumpblock-Einträgen.

```
        txt_ou: equ  #BB5D              ;jblock output routine
        disabl: equ  21                 ;screen disable char
FE15    new:    cp   disable            ;routine entry point
C8              ret  z                  ;ignore disable
        ;now switch old entry into jumpblock and call it
CD2529          call copy               ;switch entry
CD5DBB          call txt_out            ;send character
CD2529          call copy               ;copy the entry back
C9              ret
        ;
F5      copy:   push af                 ;must preserve all regs
C5              push bc
D5              push de
E5              push hl
0603            ld   b,3                ;no of bytes in entry
215DBB          ld   hl,txt_out         ;old entry
113F29          ld   de,txt_cpy         ;copy area
4E      cloop:  ld   c,(hl)             ;read current entry
1A              ld   a,(de)             ;and its replacement
77              ld   (hl),a             ;substitute it
79              ld   a,c
12              ld   (de),a             ;and save the old one
23              inc  hl                 ;point to next byte
13              inc  de
10F7            djnz cloop              ;do all three
E1              pop  hl
D1              pop  de
C1              pop  bc
F1              pop  af
C9              ret
        ; new jumpblock entry is store below until required
C3      txt_cp: defb #c3                ;code for jmp
1829            defw new                ;to substitute routine
```

Wichtig ist, daß das neue Modul wie die ursprüngliche Firmware-Routine reagiert. Alle Parameter, die an die Routine gehen und von dort empfangen werden, müssen daher die gleiche Definition haben wie in der Originalroutine. Unser Beispielprogramm sichert deshalb die zurückgelieferten Register vor Ablauf des Kopiervorganges.

Die neuen Routinen sollten unterhalb von HIMEM liegen. Die Aufteilung des Arbeitsspeichers zeigt, daß die unteren drei 16K-RAM-Blöcke für BASIC zur Verfügung stehen. Die untere Grenze ist fest eingebaut, während HIMEM die obere Grenze bestimmt. BASIC-Programme „wachsen" von der Untergrenze aus in Richtung HIMEM.

Da der für Maschinencode reservierte Platz nicht von BASIC überschrieben werden darf, muß HIMEM heruntergesetzt werden. HIMEM sollte jedoch nur um die Länge der Routine (plus seines Datenbereiches) herabgesetzt werden, damit BASIC nicht zuviel Platz weggenommen wird. Die Routine wird dann in den auf diese Weise „befreiten" Platz geladen.

Der Inhalt des Speicherbereiches zwischen #0000 und #003F ist in RAM und ROM identisch. Damit ist die Ausführung von RST unabhängig davon, ob das untere ROM ein- oder ausgeschaltet ist. Der Z80 verfügt über acht Neustart-Befehle, die hauptsächlich für die Speicherverwaltung eingesetzt werden. Die Tabelle zeigt ihre Funktionen.

Hexadresse	Funktion
0000 RST 0	Reset (Wie beim Anschalten des Gerätes)
0008 RST 1	Sprung auf eine Routine der unteren 16K. Die beiden Bytes hinter RST werden als 14-Bit-Adressen (0000 bis 3FFF) interpretiert. Bit 15 gesetzt = oberes ROM abgeschaltet, Bit 14 gesetzt = unteres ROM abgeschaltet.
0010 RST 2	Aufruf eines Zusatz-ROMs – hinter RST stehen zwei Bytes. Bit 14 und 15 zeigen auf eins der Zusatz-ROMs; Bit 0 bis 13 enthalten ein Offset, das auf #C000 addiert wird und so die Adresse der gewünschten Routine angibt.
0018 RST 3	Ruft eine beliebige Routine im ROM oder RAM auf. RST plus zwei Bytes zeigen auf eine Drei-Byte-Adresse. Dabei geben Byte 0/1 die Adresse an, während Byte 2 das ROM bezeichnet: #00–#FB = Nummer des Zusatz-ROM – oberes ROM an, unteres aus; #FC = oberes und unteres ROM an; #FD = oberes RAM an, unteres aus; #FE = oberes ROM aus, unteres an; #FF = beide ROMs aus.
0020 RST 4	Das Byte, auf das HL zeigt, aus dem RAM lesen – oberes und unteres ROM abschalten. Beim Rücksprung enthält A das gelesene Byte.
0028 RST 5	Firmware-Aufruf. Schaltet das untere ROM an und aktiviert die Routine, auf die die beiden Bytes hinter RST zeigen.
0030 RST 6	Neustart durch den Anwender. Bytes #30 bis #37 sind unbelegt.
0038 RST 7	Einsprungspunkt der Interrupts. Sollte möglichst nicht eingesetzt werden. Wenn Sie Interrupts des Modus 1 verwenden wollen, können Sie mit „Ereignissen" der Firmware arbeiten, auf die wir später genauer eingehen.

Einen Speicherblock in die Datei schreiben

Der folgende Ablauf schreibt einen Speicherblock auf Cassette oder Diskette:

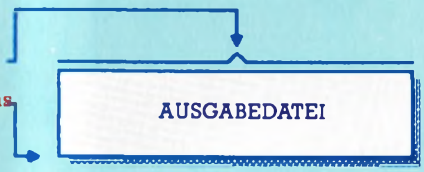
1) Ausgabedatei eröffnen

B = Länge des Dateinamens

HL = Adresse des Dateinamens

DE = Adresse des 2K-Buffers

CALL CAS OUT OPEN



2) Daten schreiben

HL = Adresse der Ausgabedaten

DE = Länge der Daten

BC = Eintragsadresse des Kopfsatzes

A = Dateityp für den Kopfsatz

CALL CAS OUT DIRECT

3) Ausgabedatei schließen

CALL CAS OUT CLOSE

Format des Kopfsatzes

Bei der Eröffnung einer Datei mit CAS IN OPEN wird der Kopfsatz der Datei ins RAM geschrieben und seine Anfangsadresse an HL zurückgegeben. Weiterhin enthält DE die ursprüngliche Speicheradresse der Daten (wie im Kopfsatz eingetragen). In BC ist die logische Dateilänge gespeichert und in A der Dateityp. Die Informationen des Dateityp-Bytes haben bitweise Bedeutung (siehe unten).

Der Dateityp wird von mehreren Feldern definiert:

Bit 0	Bit 1–3	Bit 4–7
Dateischutz	Dateiinhalt	Version
0 = Geschützt 1 = Nicht geschützt	0 = BASIC 2 = Binär 4 = Bildschirm 6 = ASCII 8 = nicht zugeteilt	16 = ASCII 32 nicht zugeteilt

Gut und sicher

Mit dem integrierten Cassettendeck des Schneider CPC 464 und der Diskettenstation des CPC 664 lassen sich Dateien leicht zwischen diesen beiden Maschinen austauschen.

Der Erfolg des Schneider CPC 464 wurde wesentlich von dem integrierten Cassettendeck mitbestimmt, das Daten selbst bei hoher Schreibgeschwindigkeit sicher speichern kann. Der CPC 664 stellte dann statt des Recorders eine Diskettenstation mit noch höherer Speichergeschwindigkeit zur Verfügung. Da aber auch die beste Hardware ohne entsprechende Software wertlos ist, untersuchen wir, welche Firmware-Routinen dem Programmierer dafür zur Verfügung stehen.

Die Firmware-Routinen lassen sich in drei Bereiche unterteilen: die allgemeine Dateiübertragung, die Steuerung des Bandsystems und schließlich die Diskettenroutinen.

Jede Datei enthält zwei separate Informationsbereiche: den Kopfsatz und die Daten. Im Kopfsatz sind alle Informationen gespeichert, die die Firmware zur Bearbeitung der Datei braucht – zum Beispiel der Dateityp und die Dateilänge. Auf den Kopfsatz folgt der eigentliche Dateiinhalt – die Daten.

Wir werden uns zunächst mit dem „Cassettenmanager" beschäftigen. Der Begriff ist etwas irreführend, da die gleichen Routinen auch für Diskettendateien eingesetzt werden, auf die wir später eingehen.

Der Cassettenmanager

Der Cassettenmanager arbeitet mit zwei „Strömen" (für Ein- und Ausgabe), die beide gleichzeitig aktiv sein können. Bei einem Bandgerät läßt sich diese Fähigkeit kaum einsetzen, bei einer Diskettenstation ist sie jedoch außerordentlich praktisch. Vor ihrem Einsatz müssen die Ströme initialisiert werden. Dabei wird dem entsprechenden Strom ein Name und ein Arbeitsbereich (Buffer) zugewiesen. Da nur jeweils eine Ein- und eine Ausgabedatei aktiv sein kann, läßt sich der Ausgabestrom nicht als zweiter Eingabestrom verwenden. Die Datenübertragungsroutinen sind in einer speziellen Tabelle eingetragen.

Daten lassen sich auf zwei Arten zu einer Datei schicken oder von dort empfangen: per Block oder per Zeichen. Bei der Zeichenüber-

tragung benutzt die Firmware den (dem Strom zugeordneten) Buffer als Zwischenspeicher. Das bedeutet, daß die Daten blockweise zum Band (oder zur Diskette) gesandt oder von dort empfangen werden, während das Programm sie byteweise bearbeitet. Natürlich ist ein Programm, das den Speicher zeichenweise anspricht, sehr langsam.

Die beiden Übertragungsarten schließen sich gegenseitig aus. Wenn eine Datei einmal auf eine bestimmte Art angesprochen wurde, kann nicht mehr auf die andere Art umgeschaltet werden.

Bei der Blockübertragung (CAS IN DIRECT und CAS OUT DIRECT) ist der Buffer nicht nötig, da die Daten direkt aus dem Speicher gelesen bzw. dort abgelegt werden.

Die Routinen CAS IN CLOSE und CAS OUT CLOSE sind besonders wichtig, da sie die Firmware informieren, daß der Strom geschlossen wurde und für eine neue Datei zur Verfügung steht. Wenn nach einer Zeichenausgabe kein CAS OUT CLOSE folgt, kann es vorkommen, daß der Buffer-Inhalt nicht auf das Speichermedium übertragen wird. CAS OUT CLOSE zwingt die Firmware, auch teilweise gefüllte Datenblöcke zu senden, und stellt so sicher, daß alle Daten geschrieben werden.

Unser erstes Bild zeigt, wie Datenblöcke in den Speicher geschrieben werden. Zunächst eröffnet CAS OUT OPEN die Datei und reserviert einen Namen und einen Ausgabestrom dafür. Danach werden die Daten mit CAS OUT DIRECT ausgegeben, und die Datei wird mit CAS OUT CLOSE wieder geschlossen.

In den Kopfsätzen speichert der Cassettenmanager alle wichtigen Dateiinformationen. Ein Lesevorgang holt diese Information aus dem Dateianfang, während ein Schreibvorgang den Kopfsatz in den Buffer schreibt und dann auf den Dateianfang überträgt.

Auf der Firmware-Ebene hat der Parameter **Dateityp** keine Auswirkung – jede Datei kann unabhängig von ihrem Typ gelesen oder geschrieben werden. Wichtig wird der Typ erst bei einem Programm auf höherer Ebene.

Die **Datenadresse** ist als die Speicherstelle definiert, von der aus die Datei ursprünglich geholt wurde. Dies hat jedoch nur dann Bedeutung, wenn die Datei direkt aus dem Speicher

Routinen zur Datenspeicherung

Cassetten-System:

$BC65 CAS INITIALISE
Stellt sicher, daß alle Standardeinstellungen gesetzt sind (Ein- und Ausgabeströme geschlossen, die Meldungsroutine eingeschaltet).

$BC6B CAS NOISY
Schaltet vom Cassettenmanager gelieferte Meldungen ein/aus.

$BC77 CAS IN OPEN
Initialisiert eine Datei zum Lesen

$BC80 CAS IN CHAR
Liest einzelnes Zeichen aus Eingabedatei

$BC83 CAS IN DIRECT
Liest ganze Datei in den Speicher

$BC89 CAS TEST EOF
Prüft auf das Ende einer Eingabedatei

$BC7A CAS IN CLOSE
Schließt die aktuelle Eingabedatei

$BC7D CAS IN ABANDON
Schließt aktuelle Eingabedatei, selbst wenn sie noch nicht vollständig geladen wurde.

$BC8C CAS OUT OPEN
Initialisiert eine Datei zum Schreiben

$BC95 CAS OUT CHAR
Sendet einzelnes Zeichen zur Ausgabedatei

$BC98 CAS OUT DIRECT
Schreibt Speicherblock in die Ausgabedatei

$BC8F CAS OUT CLOSE
Schließt Datei nach Ende des Schreibvorganges

$BC92 CAS OUT ABANDON
Schließt die Datei und ignoriert alle nicht geschriebenen Daten

$BC9B CAS CATALOG
Legt einen Dateikatalog an

Disketten – System:

$BE80 SET MESSAGE
Meldungen der Diskettenroutine ein- und ausschalten

$BE83 SETUP DISK
Initialisiert die Zeitparameter des Laufwerks (Schrittgeschwindigkeit, Verzögerung von Load/unload und Timeout Ein/Aus)

$BE86 SELECT FORMAT
Diskettenformat wählen

$BE89 READ SECTOR
Liest einen Diskettensektor

$BE8C WRITE SECTOR
Schreibt einen Diskettensektor

$BE8F FORMAT TRACK
Formatiert eine Spur

$BE92 MOVE TRACK
Stellt Schreib-Lesekopf auf angegebene Spur

$BE95 GET DR STATUS
Liefert ein Byte, das den Status des aktuellen Laufwerks anzeigt

$BE98 SET RETRY COUNT
Setzt die Zahl der Wiederholungsversuche bei Fehlern.

Cassetten- und Diskettenabläufe werden mit Firmware-Routinen ausgeführt. Ein- und Ausgabedateien können gleichzeitig eröffnet sein. Die Verwaltungsbefehle von Cassetten- und Diskettendateien sind nahezu identisch. Dateien lassen sich direkt aus dem RAM abrufen und dort speichern (CAS IN/OUT DIRECT) oder zeichenweise über einen 2K-Buffer (CAS IN/OUT CHAR). Nach Eröffnung einer Datei kann die Ein- und Ausgabeart nicht mehr verändert werden.

genommen wird – bei der Zeichenausgabe steht hier die Adresse des Ausgabespeichers. **Logische Länge** ist die Gesamtzahl der Datenbytes einer Datei, ohne Kopfsatz oder Zeichen, die zum Füllen des letzten Datenblocks eingesetzt wurden.

Einsprungadresse ist eine Speicherstelle, von der aus Maschinencodeprogramme vom BASIC aus automatisch gestartet werden. Die anderen Felder beziehen sich auf die Bearbeitung von Cassettendateien.

Die Firmware führt automatisch eine Fehlerprüfung durch, die mit einem CRC-Code arbeitet. Dank der umsichtigen Systemprogrammierung sind Fehler jedoch selten. Die Fehler werden von der Firmware angezeigt.

Um Lesefehler leichter abfangen zu können, werden Diskettendateien nicht als Ganzes geschrieben, sondern in 2K-Blöcken. Da jeder Block als separate Einheit behandelt wird, lassen sich fehlerhafte Blöcke leicht nochmals laden, ohne daß fehlerlose Blöcke wiederholt werden müssen. Die Blockstruktur verlangsamt zwar die Übertragung, macht die Speicherung aber sehr sicher. Die Schreibgeschwindigkeit auf Cassette läßt sich per Software über CAS SET SPEED (bei $BC68) einstellen, die Lesegeschwindigkeit wird von der Firmware automatisch festgesetzt.

Transparente Befehle

Die transparent angelegten Diskettenbefehle sind eine der praktischsten Einrichtungen des Betriebssystems. Da dadurch alle allgemeinen Dateiroutinen mit den gleichen Parametern arbeiten, laufen für Cassette geschriebene Programme ohne Änderung auch mit Disketten. Dies beruht auf der Technik des „Jumpblock-Patchens", die wir bereits beschrieben haben. Hier einige Hinweise, die das Verständnis der Patch-Technik erleichtern.

Speziell der Einsatz des Speichers will genau überlegt sein. Das Diskettensystem wird zwar von einem Zusatz-ROM gesteuert, braucht aber auch einen gewissen Arbeitsbereich im RAM. Dieser Platz liegt normalerweise oben im Speicher, kann aber auch anderen RAM-Bereichen zugeordnet werden. Bei Programmen, die mit Disketten arbeiten, ist darauf zu achten, daß dieser RAM-Bereich ($504H unter HIMEM) nicht verfügbar ist.

Weiterhin dürfen Dateinamen wegen des CP/M-Formats nicht länger als zwölf Zeichen sein. Wenn Sie sich auf acht Zeichen beschränken, vermeiden Sie Probleme, die bei der Übertragung von Cassette auf Diskette entstehen können.

Die Kopfsätze der Diskettendateien werden anders behandelt als die der Cassettendateien: Die Blockinformationen haben hier keine Bedeutung und werden ignoriert. Da der Kopfsatz mit einer Prüfsumme versehen ist, kann die Firmware außerdem leicht zwischen CP/M- und AMSDOS-Dateien unterscheiden.

Unser Beispielprogramm zeigt, wie Informationen aus dem Kopfsatz einer Datei herausgezogen werden. Die Datei wird zunächst mit dem Aufruf von CAS IN OPEN eröffnet, der automatisch im RAM einen Kopfsatz anlegt. Nachdem die entsprechende Information im Speicher ist, wird die Datei vor dem Lesen mit CAS IN ABANDON abgeschaltet. Das BASIC-Programm dient als Anwenderschnittstelle für den Maschinencode.

```
10 'File Header Reader
20 'Uses CAS IN OPEN at BC77H
30 routine=&8800: address=&8000: ON ERROR GOTO 530
40 MODE 1: MEMORY &5FFF
50 WHILE -1
60 CLS
70 LOCATE 10,10: PRINT "Tape or disc file? ";CHR$(18)
80 a$=""
90 WHILE a$()"T" AND a$()"D"
100 a$=UPPER$(INKEY$)
110 WEND
120 IF a$="T" THEN ITAPE ELSE IF a$="D" THEN IDISC
130 LOCATE 10,12: INPUT "Filename";file$
140 IF LEN(file$)=0 AND a$="D" THEN 130
150 GOSUB 420
160 CALL routine
170 buffer=PEEK(address)+256*PEEK(address+1)
180 LOCATE 20,12: ff=0
190 FOR name=0 TO 7-8*(t$="T")
200 char=PEEK(buffer+name)
210 IF (char(32 OR char)126) AND char)0  THEN  ff=1: GO
TO 230
220 PRINT CHR$(char);
230 NEXT
240 IF ff=1 THEN PRINT CHR$(8);"No filename";
250 PRINT
260 'test filetype
270 LOCATE 5,16: PRINT "Filetype: ": LOCATE 20,16
280 type=PEEK(buffer+18)
290 IF (type AND 1)=1 THEN PRINT "Protected ";
300 file = type AND &FE
310 IF file=0 THEN PRINT "Basic"
320 IF file=2 THEN PRINT "Binary"
330 IF file=6 THEN PRINT "ASCII"
340 IF file)6 THEN PRINT "Unknown"
350 length=PEEK(buffer+24)+256*PEEK(buffer+25)
360 LOCATE 5,18: PRINT "Data length:":LOCATE 20,18: PRI
NT HEX$(length,4)
370 entry=PEEK(buffer+26)+256*PEEK(buffer+27)
380 LOCATE 5,20: PRINT "Entry address:": LOCATE 20,20:
PRINT HEX$(entry,4)
390 LOCATE 9,25:PRINT "Press a key to continue"
400 WHILE INKEY$="":WEND
410 WEND
420 RESTORE 520: READ byte: off=0
430 WHILE byte()999
440 POKE routine+off,byte
450 off=off+1:READ byte
460 WEND
470 POKE routine+1,LEN(file$)
480 FOR char=1 TO LEN(file$)
490 POKE routine+&100+char-1,ASC(MID$(file$,char,1))
500 NEXT
510 RETURN
520 DATA &06,&00,&21,&00,&89,&11,&00,&60,&cd,&77,&bc,&7
d,&32,&0,&80,&7c,&32,&01,&80,&cd,&7d,&bc,&c9,999
530 'error trap
540 IF ERL=80 THEN a$="":RESTORE
550 STOP
```

Motorolas Meisterstück

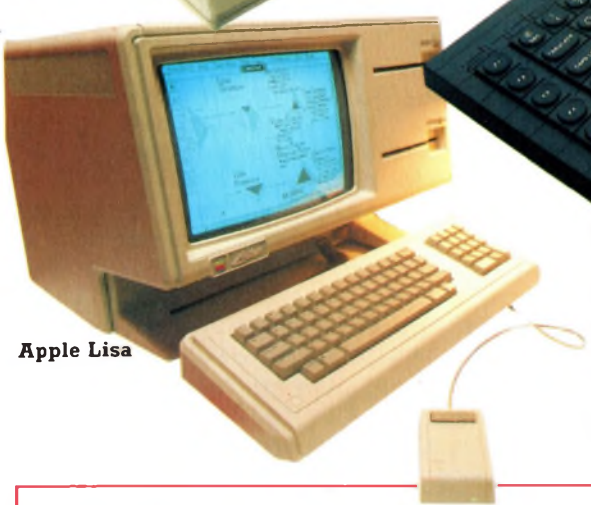
Der erweiterte Adreßbereich des 68000-Microprozessors reicht an viele Maschinen der Groß-EDV heran. In unserer Serie über den 68000 zeigen wir zunächst die Bereiche der Prozessorarchitektur, die für die Programmierung wichtig sind.

Durch die 68000-Serie erhalten die modernen Micros die Möglichkeiten von Minicomputern. Viele Hersteller haben diesen Vorteil erkannt.

Atari 520 ST

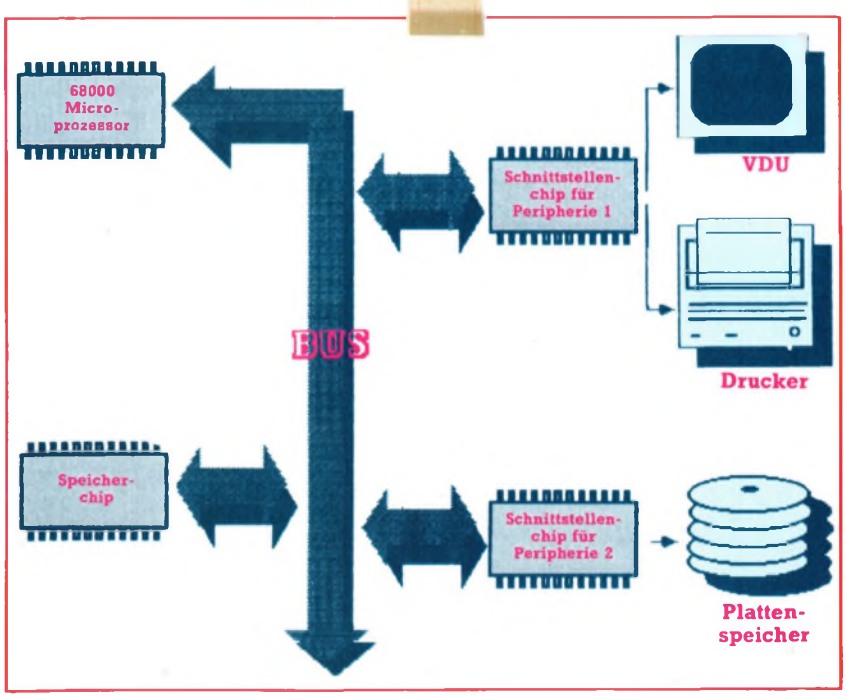
Apple Macintosh

Sinclair QL



Apple Lisa

Die Befehle und Daten eines Einplatinencomputers werden über einen Bus aus dem Speicher geholt und dann verarbeitet. Danach wird das Ergebnis wieder auf den Bus gesetzt, der es zu Peripheriegeräten sendet.

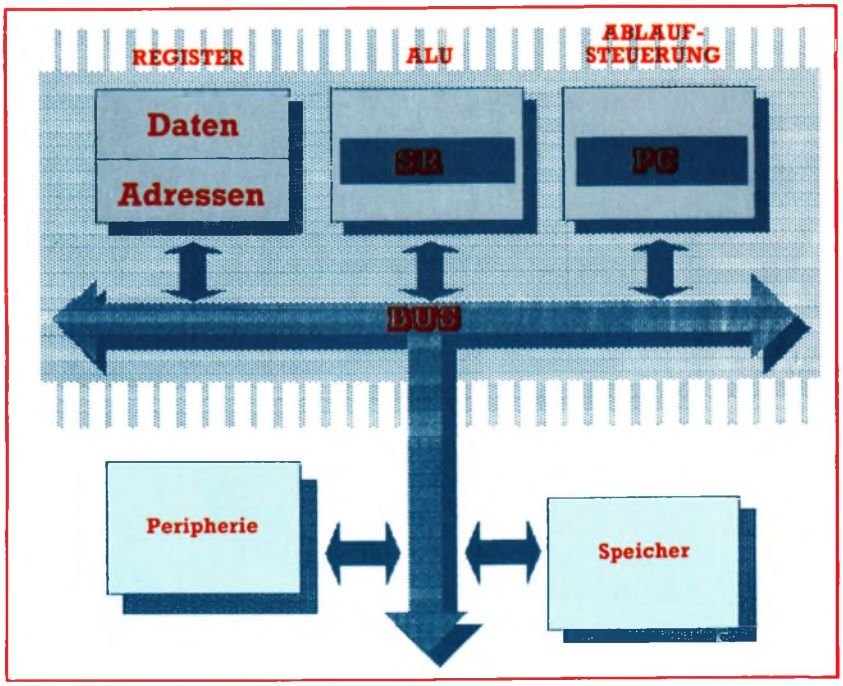


Der 68000 entwickelte sich aus der erfolgreichen 6800-Serie von Motorola, zu der auch der 6809-Microprozessor gehört. Mit der Familie der 68000-Chips baute Motorola seinen Erfolg weiter aus. Bei dem Prozessor stand die neue VLSI-Technik (Very Large Scale Integration) Pate, mit der mehr als 100 000 Logikelemente auf einem etwa fünf Zentimeter langen und mit 64 Kontakten versehenen Chip untergebracht wurden. Die Adressiermöglichkeiten reichen an die des DEC PDP-11 Minicomputers heran, wobei der 68000 den Komfort und die Flexibilität eines Befehlssatzes der Groß-EDV bietet.

Die Adreß- und Befehlsmöglichkeiten werden wir uns in späteren Folgen genauer ansehen. Zunächst wollen wir uns einen Überblick über die Architektur des 68000 verschaffen, um ein Modell für unsere Programmabläufe zu haben. Da wir nicht auf alle Aspekte des Chips eingehen können, beschränkt sich unser Modell nur auf die Bauteile, die für unsere Beispielprogramme wichtig sind. So brauchen wir beispielsweise für die Addition zweier Zahlen nicht die Arbeitsweise der Flip-Flop-Elemente zu verstehen, sondern müssen uns lediglich mit der Registergröße und dem Befehl ADD beschäftigen. In unserem Modell beschränken wir uns daher auf die Funktionen, mit denen der Microcomputer programmiert wird.

Einplatinensystem

Wenn wir die Funktionen der Kontakte beim 68000 untersuchen, stellen wir fest, daß damit digitale Busse gesteuert werden, über die der Prozessor mit seiner „elektronischen Umgebung" Daten austauscht. Mit den Bussen läßt sich ein vollständiges Microcomputersystem aufbauen. Unser erstes Bild zeigt ein typisches Einplatinensystem, in dem sich der Prozessor den nächsten Befehl aus dem Speicher holt, ihn ausführt und schließlich – falls gewünscht – ein Zeichen zum Schnittstellenchip sendet,



Der innere Aufbau des 68000 läßt sich in drei Hauptbereiche unterteilen: die Daten- und Adreßregister, die Arithmetik- und Logikeinheit und die Ablaufsteuerung. Jede Einheit kann über einen digitalen Datenbus Informationen austauschen.

der das Zeichen dann direkt auf den Bildschirm überträgt.

Dieser Hardwareaufbau ist Grundlage unseres Modells. Dabei ist zwar der zentrale Datenbus die wichtigste Komponente, doch spielen auch die internen Register des Microprozessors eine wesentliche Rolle.

Unser zweites Bild zeigt den Chip aus der Sicht des Programmierers. Wir werden uns die drei logisch voneinander getrennten Bereiche einmal genauer ansehen:

Die Register sind – wie auch in anderen Prozessoren – Arrays mit Speicherstellen, die zur Kurzspeicherung von Daten oder Zwischenergebnissen eingesetzt werden. Der 68000 verfügt über acht Daten- und acht Adreßregister, die je eine Breite von 32 Bit haben. Durch die Aufteilung in zwei Bereiche werden Maschinenabläufe und Programmierung doch schon sehr vereinfacht.

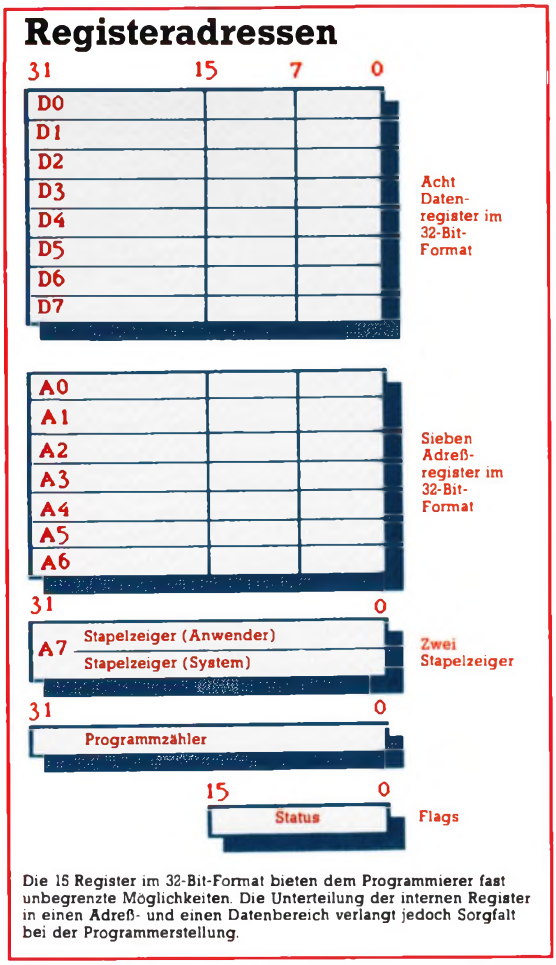
Wenn Sie beispielsweise den Inhalt einer Speicherstelle (deren Adresse sich bereits in einem der internen Register befindet) laden wollen, um einen arithmetischen Ablauf durchzuführen, muß zunächst der Pointer (oder die Speicheradresse) in ein Adreßregister geladen werden. Daraufhin können die Daten in einem Datenregister gespeichert werden. In der Assemblersprache wird daraus:

```
MOVE (A2),D4
```

MOVE ist der Befehl zum Übertragen (oder Kopieren) von Daten, A2 gibt als Quellenadresse den Inhalt des Adreßregisters 2 an, während D4 das Datenregister 4 als Ziel festlegt. Wenn die Adresse der Quelldaten in D4 läge, könnten wir jedoch nicht einfach

```
MOVE (D4),A2
```

schreiben, da eine Adressierung von Quelle und Ziel beim Befehl MOVE nicht zulässig ist.



Die 15 Register im 32-Bit-Format bieten dem Programmierer fast unbegrenzte Möglichkeiten. Die Unterteilung der internen Register in einen Adreß- und einen Datenbereich verlangt jedoch Sorgfalt bei der Programmerstellung.

Wichtig ist weiterhin, daß speziell die Datenregister fünf voneinander verschiedene Datentypen annehmen:

- Bit – eine Binärstelle
- BCD-Stelle – ein Nybble oder Halbbyte
- Byte – ein Acht-Bit-Zeichen
- Word – ein 16-Bit-Computerwort
- Long Word – ein 32-Bit-Computerwort.

Auf BCD-Stellen und Langwörter gehen wir später noch genauer ein, für den Augenblick wollen wir sie einfach als größere Dateneinheiten ansehen.

Das Befehlsattribut

Bei den meisten Befehlen kann der Datentyp als Befehlsattribut angegeben werden. Wenn Sie beispielsweise die Bits 0 bis 7 von D1 auf D2 kopieren wollen, schreiben Sie

```
MOVE.B D1,D2
```

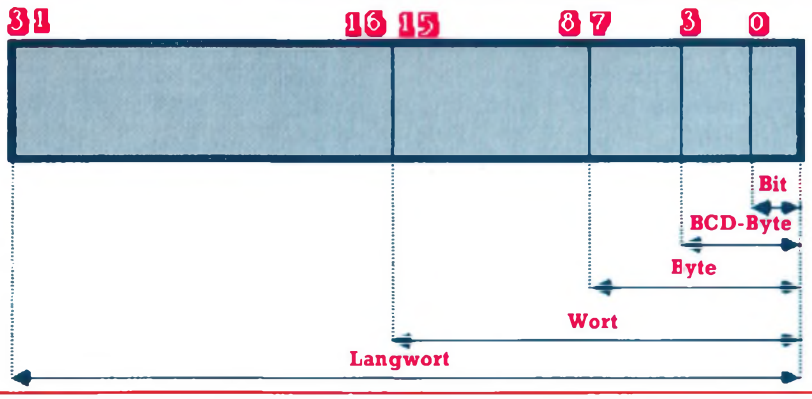
Alle 32 Bits lassen sich mit folgendem Befehl kopieren:

```
MOVE.L D1,D2
```

Das Attribut (oder der Code für die Datenlänge) eines 16-Bit-Wortes lautet .W. Dies ist die Standardeinstellung, wenn kein anderer Code angegeben wird.

Wortlängen

Der 68000 kann über Befehlserweiterungen Wörter mit fünf verschiedenen Längen adressieren und damit Daten in Gruppen von einem bis 32 Bits verarbeiten. Diese Möglichkeit gab es bei den älteren Acht-Bit-Chips nicht.



Noch ein Wort zum Registersatz. Das Adreßregister A7 wird als Pointer für den Systemstack eingesetzt – seien Sie daher beim Gebrauch dieses Registers äußerst vorsichtig.

Die ALU (Arithmetik- und Logikeinheit) ist der zweite große Bereich unseres Modells. Dieser Teil führt die arithmetischen und logischen Vorgänge aus und speichert das Ergebnis im Zieloperanden. So addiert die ALU bei folgendem Befehl D1 und D2 und stellt das Ergebnis in D2:

```
ADD D1,D2
```

Auch hier können Sie im Befehl die Datenlänge angeben und nur die niederwertigen Bytes addieren:

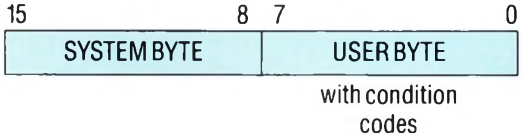
```
ADD.B D1,D2
```

In unserem Modell ist der ALU das Register SR oder Status Register beigeordnet, das Informationen über das Ergebnis des vorangegangenen Befehls liefert. So etwas wird beispielsweise für eine bedingte Verzweigung gebraucht, die vom Ergebnis des vorigen Befehls abhängt.

```
ADD D1,D2     D1 mit D2 addieren
BVS OFLOW     auf Überlauf prüfen
MOVE D1,D3    D1 auf D3 kopieren
```

Hier wird entweder zum Label OFLOW verzweigt, wenn das Überlaufbit in SR gesetzt ist, oder D1 auf D3 kopiert, wenn es nicht gesetzt wurde. Der Befehl BVS (Verzweigung bei Überlauf, englisch: „Branch if Overflow Set") prüft, ob ein Überlauf stattgefunden hat (oder Bit V des Statusregisters gesetzt ist). In diesem Beispiel könnte das Ergebnis des ADD-Befehls das Setzen des V-Bits verursachen. Überläufe entstehen, wenn das Ergebnis eines arithmetischen Vorgangs nicht mehr in die Wortlänge der Operanden paßt.

Das Statusregister besteht aus 16 Bits, und seine Bytes werden vom System wie folgt aufgeteilt:



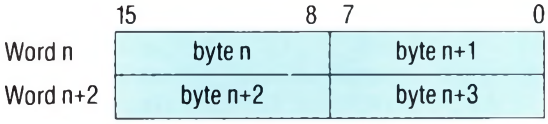
Die einzelnen Bits zeigen das Ergebnis der vorangegangenen Befehle an – wir gehen später darauf genauer ein.

Adressierung

Der Bereich für Ablaufsteuerung enthält den Programmzähler und ein Register, in dem die nächste Adresse steht, die aus dem Speicher geholt werden soll. Nach dem Laden wird ein Befehl erst einmal decodiert, um festzustellen, welche Art Ablauf die ALU ausführen soll, und wo die Quell- und Zieloperanden liegen.

Der Programmzähler ist zwar 32 Bit lang, aber nur mit 24 Kontakten an den Bus angeschlossen. Doch selbst mit 24 Bits steht ein riesiger Adreßbereich bis zu Hex FFFFFF Bytes zur Verfügung. Jede Hexadezimalstelle entspricht vier binären Bits, so daß die 24 Bits bis zu 16 777 216 Bytes direkt adressieren können. Da jedoch alle Befehle nur mit geraden Adressen beginnen dürfen, kommt ein Adreßbereich von 8 388 608 Computerwörtern der Wirklichkeit näher.

Interessant ist auch die Anordnung der Daten im Speicher. So lassen sich mit dem 68000 einzelne Bytes ansprechen, und die Adressiermethode unterscheidet sich erheblich von den bereits besprochenen Adressierungsarten. Die folgende Zeichnung macht deutlich, wie die Speicheradressierung des 68000 das höchstwertige Byte eines Wortes liefert.



Einige Befehle wirken direkt auf den Programmzähler und veranlassen so eine Programmverzweigung; entweder ohne Bedingung wie bei BRA ZURUECK (BRA – englisch „Branch Always" – bedeutet „Unbedingte Verzweigung" auf die Adresse, die das Label ZURUECK darstellt) oder bedingt wie in unserem letzten Beispiel BVS OFLOW, in dem die Verzweigung von dem Setzen des V-Bits im PS abhängig ist. Bei jeder Verzweigung oder Änderung des Programmflusses wird der Pro-

grammzähler auf die Adresse des nächsten Befehls gesetzt, der ausgeführt werden soll.

Computerhersteller preisen oft die große Zahl von Befehlen an, die auf ihren Maschinen zur Verfügung stehen. Diese Befehlsvielfalt kann jedoch sehr verwirren, wenn für jeden Operandentyp oder Adreßmodus ein anderer Befehl verwandt werden muß. Dies trifft besonders auf die älteren Acht-Bit-Prozessoren zu. Der 8085 besitzt beispielsweise 63 verschiedene MOV-Befehle, die die Register auf unterschiedliche Weise ansprechen.

Viel wichtiger als die Zahl der Befehlscodes ist der damit erfaßte Bereich, die Vielfalt der unterstützten Datenobjekte und die Flexibilität der Adreßarten.

Flexibler Befehlssatz

Motorola bietet einen außerordentlich flexiblen Befehlssatz für Datenobjekte. Die meisten Befehle sind für die Adressierung von Bytes, Wörtern und Langwörtern ausgelegt. Sie können binäre Multiplikation, Division und BCD-Arithmetik ausführen und verfügen über die übliche Befehlspalette für Logikabläufe, Programmsteuerung und das Kopieren von Daten.

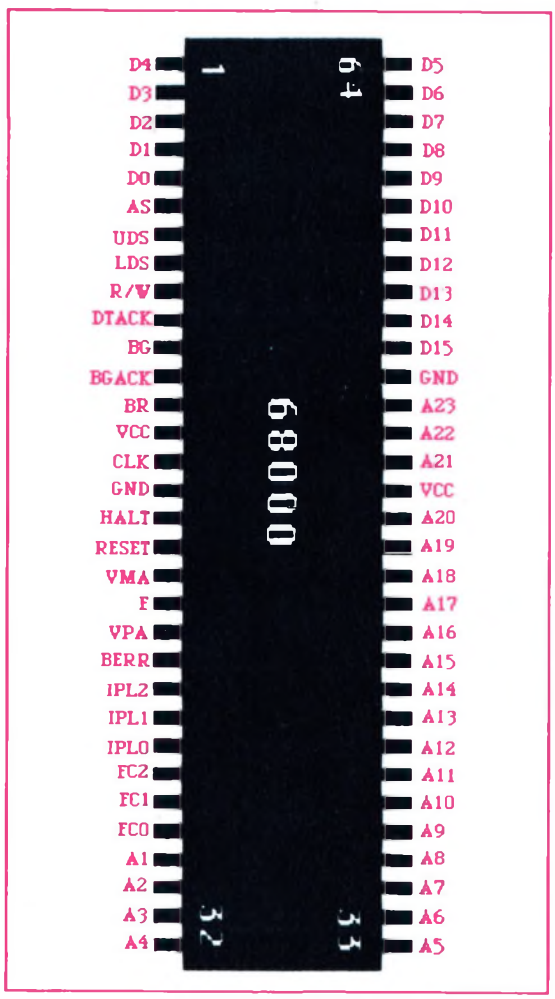
Die vielen Adreßarten lassen sich jedoch nicht in allen Befehlen einsetzen. enn der Zieloperand des MOVE-Befehls beispielsweise ein Adreßregister ist, können Sie nicht einfach MOVE D3,A6 schreiben, sondern müssen MOVEA oder LEA (MOVE A D3,A6) verwenden. Hier wurden einige Adreßmöglichkeiten zugunsten einer großen Zahl der möglichen Datenobjekte geopfert.

Im Vergleich zu den Befehlssätzen der Acht-Bit-Micros (und selbst einigen Computern der Mini- und Groß-EDV) hat der 68000 einen ungewöhnlich breiten Anwendungsbereich, kann eine Vielzahl von Datenobjekten verarbeiten, enttäuscht aber mit der wenig flexiblen Adressierung. Dieser Nachteil wird durch die Kompaktheit einiger Befehle (auch „Quick"-Befehle genannt) teilweise wieder ausgeglichen. Bei Quick-Befehlen ist der gesamte Ablauf in einem Maschinenwort enthalten. Wenn Sie zum Beispiel die Zahl 25 in ein Datenregister laden wollen, genügt

```
MOVEQ #25,D3
```

wonach 25 in D3 gespeichert ist. Der gesamte Befehl inklusive der Konstanten 25 belegt nur ein Wort.

In der nächsten Folge sehen wir uns Einsatz und Struktur der Adreßarten genauer an und geben einige Programmbeispiele.



Das Bild zeigt die Kontakte des 68000. Beachten Sie die 24 Adreßleitungen und die 15 Datenleitungen. Im Gegensatz zu seinem nahen „Verwandten", dem 68008 (der nur einen Acht-Bit-Adreßbus besitzt), kann der 68000 Speicheradressen von Hex 000000 bis FFFFFF ansprechen. Der nächstgrößere Prozessor dieser Serie – der 68020 – verfügt über einen vollständigen 32-Bit-Bus. Der 68000 ist das perfekte Beispiel eines modernen Microprozessors – der Chip mit seinen mehr als 100 000 Logikschaltungen besitzt die Verarbeitungsmöglichkeiten eines Minicomputers.

Im rechten Moment

Der 16-Bit-Prozessor 68000 erschien genau zu dem Zeitpunkt, als viele Hersteller eine neue Maschinengeneration entwarfen. Unterschiedliche Versionen von ihm wurden in den Sinclair QL, den Apple Macintosh und die ST-Reihe von Atari eingebaut.

Der Prozessor ist einfach zu programmieren: In den Chip wurden siebzehn 32-Bit-Register, ein 16-Bit-Datenbus, ein 24-Bit-Adreßbus und der Befehlssatz eingebaut.

Trotz der beeindruckenden Fähigkeiten dieser Geräte ist es nicht ganz einfach, das Potential des 68000 voll zu entfalten. Der 68000 ist der restlichen Chip-Technologie so weit voraus, daß viele Computerhersteller überhaupt nur einen Bruchteil seiner eigentlichen Fähigkeiten einsetzen. Ein extremes Beispiel ist der Sinclair QL, der mit der 68008-Version des Chips und damit auch nur mit einem Acht-Bit-Datenbus arbeitet. Viele Anwendungen des QL sind deshalb kaum schneller als bei den Acht-Bit-Vorläufern.

Die Zukunft ist der 68010, der den 16-Bit-Adreßbus beibehält, aber einen virtuellen Speicher bietet. Das bedeutet, daß Programme mit logischen (virtuellen) Adressen arbeiten können, ohne sich darum kümmern zu müssen, ob diese Adressen auch physisch vorhanden sind. Damit läßt sich der gesamte RAM-Bereich einsetzen, ohne in das Gerät eingebaut sein zu müssen. Ein weiterer Chip von Motorola, der 68020, wird als 32-Bit-Computer angepriesen, der 97 % der Fähigkeiten einer Groß-EDV-Anlage besitzt.

Kurze Unterbrechung

Die Zeitsteuerung ist eine der wichtigsten Betriebssystemkomponenten. Wie die meisten anderen Heimcomputer arbeitet auch der Schneider CPC mit Interrupts und Ereignissen.

Die Zeitsteuerung des Betriebssystems ist für Abläufe zuständig, die zu bestimmten Zeitpunkten ausgelöst werden müssen. Auf den Schneider-Geräten arbeitet diese Zeitsteuerung nach zwei verschiedenen Methoden: mit Hardware-„Interrupts" und softwaregesteuerten „Ereignissen".

Interruptsignale werden außerhalb des Z80-Microprozessors erzeugt. Sie unterbrechen die laufende Programmausführung und lösen bestimmte Abläufe aus.

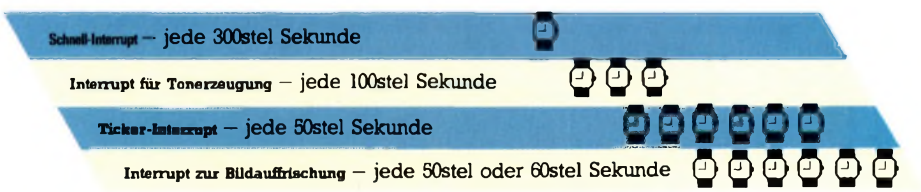
Der Z80-Prozessor des CPC wurde für Interrupts des Modus 1 programmiert. Dabei wird die Programmsteuerung bei Empfang eines Interrupts an die Interrupt-Bearbeitungsroutine (bei $0038) übergeben. Theoretisch kann jedes Gerät jederzeit Interrupts erzeugen und so die Aufmerksamkeit des Prozessors auf sich lenken. Auf Schneider-Geräten ohne Erweiterung gibt es jedoch nur den Typ des „Timer-Interrupts".

Der Timer-Interrupt wird von der ULA jede Dreihundertstelsekunde direkt mit der Systemuhr (4 MHz Taktfrequenz) erzeugt. Die interne Zeitsteuerung (die beispielsweise von den BASIC-Befehlen AFTER und EVERY benötigt wird) ist eine Funktion dieser Interruptart.

Timer-Interrupts werden direkt von den „Kernroutinen" der Firmware bearbeitet, die je nach Interruptebene (siehe Bild) weitere Firmwareroutinen aktiviert. Hier die vier Signale im einzelnen:

- Der „Schnell-Interrupt" tritt auf, wenn Kernroutinen einen Timer-Interrupt empfangen.
- Der „Interrupt für Tonerzeugung" tritt nach jedem dritten Timer-Interrupt ein und wird von der Firmware zur Synchronisierung der Datenübertragung des Klang-Chips eingesetzt. Der Anwender kann nur indirekt (durch Zählen der Schnell-Interrupts) feststellen, wann diese Art von Interrupt eintritt.
- Der „Ticker-Interrupt" tritt nach jedem sechsten Timer-Interrupt ein. Dabei löst die Firmware eine Tastaturprüfung aus.
- Der „Interrupt zur Bildauffrischung" tritt nach jedem fünften oder sechsten Timer-Interrupt auf (je nach angeschlossenem Bildschirm) und wird für alle zeitabhängigen Anzeigevorgänge eingesetzt.

Die Firmware unterhält einen Zähler für Schnell-Interrupts, der auch dem Anwender zur Verfügung steht. Damit lassen sich zeitabhängige Schleifen anlegen, die nicht von einem Programm unterbrochen werden müssen. Der Zähler kann durch den Aufruf von KL_TIME_SET gesetzt werden (der für den Zeitgeber bestimmte Wert wird im Registerpaar HL übergeben). Der Aufruf von KL_TIME_PLEASE liefert jederzeit in HL den aktuellen Zählerwert. Unser Programmbeispiel zeigt den Einsatz dieser beiden Routinen. Das Modul fragt während des angegebenen Zeitraumes die Tastatur ab und löst dann einen Rücksprung aus.



Der Prozessor der Schneider-Geräte kann das Betriebssystem auf vier Arten unterbrechen. Die „Timer-Interrupts" treten in regelmäßigen Abständen auf und synchronisieren die normalen OS-Funktionen. Sie steuern die Tastaturabfrage, die Bildschirmdarstellung und die Datenübertragung des Tonerzeugungschips.

Sie können aber auch direkt mit Timer-Interrupts arbeiten, wenn Sie den Einsprungspunkt RST 7 bei $0048 patchen. Der Patchcode sollte zuerst den bestehenden Drei-Byte-Eintrag kopieren und dort den neuen Eintrag (normalerweise ein Sprung auf eine Speicheradresse) einsetzen. Dieser neue Eintrag muß verschiebbar sein, damit das Modul auch dann funktioniert, wenn eine zweite Routine ihn wiederum an eine andere Stelle kopiert.

Die neue Routine kann einen beliebigen Ablauf ausführen (z. B. eine Uhr aktualisieren) und sollte danach aber die ursprüngliche Routine mit einem Sprung auf deren neue Adresse aktivieren. Dieser Ablauf stellt sicher, daß zeitabhängige Interruptfunktionen, wie etwa Tastaturabfragen, auch weiterhin ausgeführt werden. In früheren Folgen hatten wir schon beschrieben, wie sich „Code-Keile" in Interruptroutinen einsetzen lassen.

Bisher gingen wir davon aus, daß die ULA die Interrupts veranlaßt. Interrupts werden aber auch von externen Peripheriegeräten erzeugt. So könnte beispielsweise eine serielle Schnittstelle nach jedem empfangenen Zeichen einen Interrupt auslösen, dessen Bearbeitungsroutine das Zeichen dort „abholt".

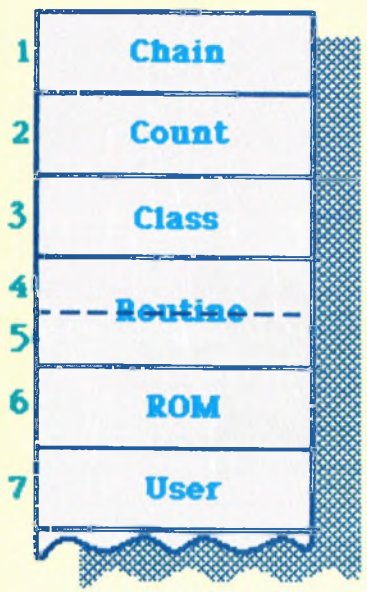
Ereignisse sind „Software-Interrupts". Mit Hilfe des Ereignisblocks behält das Betriebssystem die Übersicht über die aktivierten Ereignisse. Dabei liefern sieben aufeinanderfolgende Bytes die Adresse der Ereignisroutine, auf die sich der Block bezieht, die Zahl der Ereignisse, die noch ausgeführt werden sollen und die Ereignisklasse.

Die Kernroutine muß daher zwischen Timer-Interrupts und externen Interrupts unterscheiden können. Normalerweise ist der Z80 nach Empfang eines Unterbrechungssignals vor weiteren Interrupts abgeschirmt, damit zeitkritische Interruptroutinen nicht gestört werden. Die Kernroutinen zur Interrupt-Bearbeitung schalten den Interruptempfang jedoch wieder an. Wenn dabei das Interruptsignal zum Z80 immer noch auf Null steht (Interruptanforderung), wird ein zweiter Interrupt erzeugt. Da ein zweiter Timer-Interrupt in diesem Stadium aber nicht möglich ist, muß eine externe Interruptanforderung vorliegen – vorausgesetzt, das Interruptsignal bleibt bis zum Ende auf Null. Nach Identifizierung eines externen Interrupts wird die Steuerung an $003B übergeben. Diese Speicherstelle sollte daher mit einem Sprung auf die Interrupt-Bearbeitungsroutine gepatcht werden, die das anfragende Gerät „bedient".

Das System unterstützt „Nicht-Maskierbare Interrupts" (NMI) und Standard-Interrupts. NMIs lassen sich im Gegensatz zu Standard-Interrupts nicht abschalten. NMIs können Systemabläufe mit festem Zeitrahmen empfindlich stören und sollten daher nicht verwandt werden. So schalten beispielsweise die Firmwareroutinen zur Steuerung von Cassetten- und Diskettenlaufwerken sowie ein Teil der Klangmodule die Interrupts aus. Die Verwendung von NMIs könnte hier Chaos hervorrufen.

„Ereignisse" werden am einfachsten als „Software-Interrupts" beschrieben. Dabei führt die Firmware bestimmte Aufgaben unabhängig vom gerade ablaufenden Programm aus. Die Firmware „weiß" auch, zu welchem Zeitpunkt Ereignisse aktiviert werden sollen.

Der Ereignisblock



Systemzeiger (nicht ändern)

Ereigniszähler: Unter Null = Abgeschaltet
Null = Keine Ereignisse übrig
Über Null = Zahl der Ereignisse, die noch ausgeführt werden

Ereignisklasse: Asynchron, Synchron, Express oder Normal

Die Adresse der Ereignisroutine

Byte zur Anwahl des ROMs, in dem sich die Routine befindet

Felder, die vom Anwender definiert werden können

```
; Example of using the Kernel timer routines
;
;
SET.TIME:       EQU   £BD10      ; read the fast tick time count
GET.TIME:       EQU   £BD0D      ; set the fast tick time count
;
READ.CHAR:      EQU   £BB09      ; scan keyboard for char
PAUSE:          EQU   300        ; Pause for 300 ticks
;
                LD    HL,0       ; set time to zero
                LD    DE,0
                CALL  SET.TIME
                LD    BC, PAUSE  ; set wait period
;
; wait for a character to become available from
; the keyboard
;
WAIT:           CALL  READ.CHAR  ; character ready?
                RET   C          ; if so go back
;                                  with carry set
                CALL  GET.TIME   ; find the time
                SBC   HL,BC      ; one second?
;                                  (or greater)
                JR    C,WAIT     ; if not then retry
;
; if we get here, then routine has timed out
; go back with carry false to flag a timeout
;
                RET
;
```

Besondere Ereignisse

Zwar können Ereignisse auch externe Interrupts bearbeiten, doch eignen sie sich am besten für komplizierte Simulationsprogramme mit Vorgängen, die parallel zum „Vordergrundprogramm" ablaufen müssen. So arbeitet beispielsweise der SSA-1 Sprachsynthesizer mit Ereignissen, um Sprache von einem BASIC-Programm erzeugen zu lassen.

Ereignisse werden dem Betriebssystem als „Ereignisblock" übergeben, der mit sieben zusammenhängenden Bytes in den mittleren 32 KBytes des Arbeitsspeichers liegt. Das nebenstehende Bild zeigt den Aufbau eines Ereignisblocks. Drei Felder des Blocks, die Auskunft über den Ereignistyp und die Verarbeitungsadresse geben, gleichen sich bei allen Ereignissen.

Der „Ereigniszähler" hält fest, wie oft das Ereignis eingetreten ist. Für das Auslösen eines Ereignisses wird der Zähler auf eine positive Zahl gesetzt, die die Anzahl der Ereignisse angibt. Nach Ablauf der Ereignisroutine wird der Zähler um Eins dekrementiert. Ein auf eine negative Zahl gesetzter Zähler schaltet dann das Ereignis ab.

Die „Ereignisklasse" zeigt an, ob das Ereignis synchron oder asynchron zum Hauptprogramm laufen soll. Asynchrone Ereignisse werden unabhängig vom Hauptprogramm ausgeführt und sind für Anwendungen gedacht, die unmittelbare Reaktionen erfordern.

Synchrone Ereignisse werden bei ihrem Eintritt in eine Warteschlange gesetzt, die das Hauptprogramm je nach Bedarf in der Reihenfolge ihres Auftretens bearbeitet. Ein Ereignis läßt sich weiterhin auch auf „express" oder „normal" setzen.

In der nächsten Folge gehen wir auf die Unterschiede der Ereignisklassen genauer ein.

Richtig adressiert

In der Reihe über den 68000-Chip untersuchen wir die Adressierung der Operanden. Dieser Bereich der Programmierung hat besonders für strukturierte Datentypen wie Records und Arrays große Bedeutung.

In der letzten Folge dieser Reihe hatten wir die Adressierungsmöglichkeiten des 68000-Befehlssatzes untersucht. Dabei steht eine breite Palette von Adressierungsarten zur Verfügung, mit der sich Bytes, Computerworte oder Langworte leicht ansprechen lassen.

Der „Standardbefehl"

OPCODE Quelle, Ziel

zeigt, wie eine bestimmte Befehlsklasse die Operanden für Quelle und Ziel behandelt: Zuerst wird der Quellenoperand geholt (egal mit welchem Ablauf), dann mit dem Opcode-Vorgang bearbeitet und das Ergebnis schließlich an den Zieloperanden übergeben. So bestimmt beispielsweise der Befehl MOVEA D3,A6 daß der Inhalt von D3 (Quelle) auf A6 (Ziel) verschoben wird. Dabei gibt der Opcode MOVEA an, daß eine Adresse bewegt wird.

In diesem einfachen Beispiel wird nun die Operandenadresse direkt angegeben, doch gibt es am anderen Ende der Skala Adressierungsarten, die Operanden berechnen, indem sie den Inhalt eines Adreßregisters auf eine Offset-Zahl und ein Indexregister addieren (wie bei dem Z80-Befehl LD r,(IX+d)).

In unserem Befehlsmodell gibt es auch Opcodes mit nur einem Operanden, zum Beispiel:

OPCODE Quelle

Befehle dieser Art werden oft für Verzweigungen (wie BRA ZURUECK) eingesetzt, die nur einen Operanden brauchen, und zwar die Verzweigungsadresse (in diesem Fall „ZURUECK"). Und schließlich gibt es auch noch

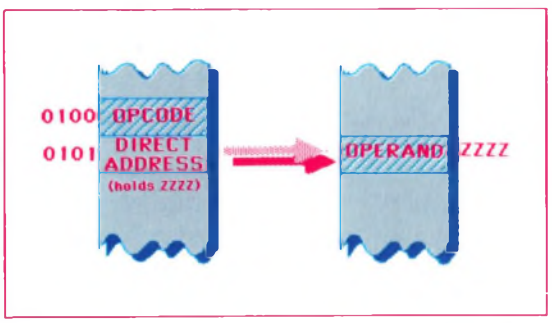
OPCODE

ohne jeden Operanden. Ein klassisches Beispiel dafür ist der Befehl NOP – die Leeranweisung. Mit diesem „Dummy"-Befehl lassen sich Programme sehr leicht im Speicher ändern oder patchen.

Bei all diesen Befehlsmodellen kann es also vorkommen, daß Adressen berechnet werden müssen. Welche Berechnungen ausgeführt werden, legt der Programmierer mit Anweisungen fest, die er aus dem Befehlssatz des 68000 und seinen Adressierungsarten auswählt.

Die meisten Computer verfügen über mindestens fünf Adressierungsarten (d. h. Methoden, die Operanden anzusprechen). Unser Bild zeigt die unterschiedlichen zwei Abläufe dieser Arten:

- **Direkte (oder absolute) Adressierung:** Dabei ist die Speicheradresse des Operanden im Befehl selbst untergebracht.
- **Indirekte (oder Pointer-)Adressierung:** Hier enthält der Befehl eine Speicheradresse, in der die Adresse des Operanden untergebracht ist.

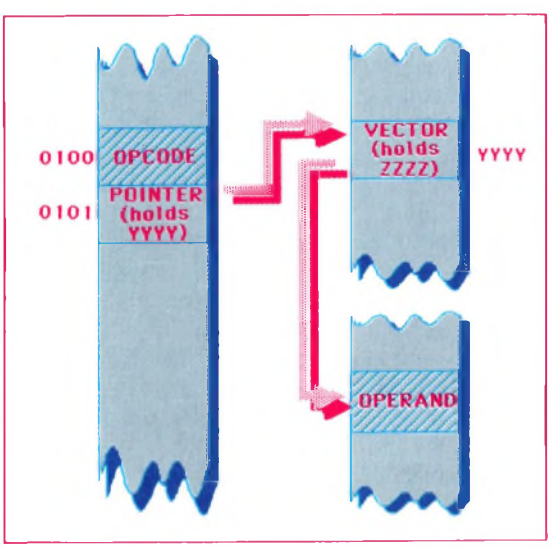


In diesem Adressiermodus folgt dem Opcode die Adresse einer Speicherstelle mit den Operandendaten.

Das Bild zeigt, daß der Opcode der direkten Adressierung direkt mit dem Operanden bei ZZZZ arbeitet. Zwar steht der Operand auch bei der indirekten Adressierung in ZZZZ, doch wird er hier über den Pointer in YYYY angesprochen.

Hier die anderen drei Adressierungsarten:

- **Unmittelbare Adressierung:** Der Befehl enthält eine Konstante. In MOVEQ #25,D3 wird die Konstante 25 unmittelbar angesprochen.
- **Registermodus:** Hier werden Register als Operanden eingesetzt und durch den Op-Code definiert. Die Operanden des Befehls MOVE D2,D4 sind die entsprechenden Datenregister D2 und D4.



Bei der indirekten Adressierung greift der Opcode über eine „Vektoradresse" auf seinen Operanden zu. Die Adresse des Vektors steht unmittelbar hinter dem Opcode, während der Vektor die Adresse des Operanden enthält. Diese Adressierungsart ist besonders praktisch, wenn die Operandenadresse erst während des Programmablaufs bekannt wird. Es muß dann nur der Inhalt des Vektors berechnet werden.

● **Implizite Adressierung:** Bei dieser Art der Adressierung liefert der Befehl selbst die Operanden. So sind beispielsweise bei RTS (Rücksprung von einer Subroutine) der Stack-Pointer und der Programmzähler die implizierten Operanden.

Interessanterweise hat der 68000 außer den bereits beschriebenen Adressierungsarten die Möglichkeit, Adressen relativ zum Programmzähler (auch „PC relativ" genannt) anzusprechen. Hier die Adressierungsarten im einzelnen:

● **Absolute Adressierung:** Kann jede beliebige Speicherstelle ansprechen. Die Adresse des Operanden steht hinter dem Befehl, zum Beispiel:

Adresse:	Code:	Label:	Befehl:	
1000	D678		ADD	DATA,D3
1002	2000			
-				
-				
-				
2000	0001	DATA	DC.W	1

Dabei wurde der Adresse $2000 der symbolische Name DATA gegeben. Der Befehl ADD mit seiner absoluten Quelladresse (DATA) und dem Ziel D3 wurde als D678 codiert, während die absolute Adresse DATA in $ 1002 (auch „Worterweiterung" genannt) gespeichert ist. Hier ein Beispiel mit nur einem Operanden:

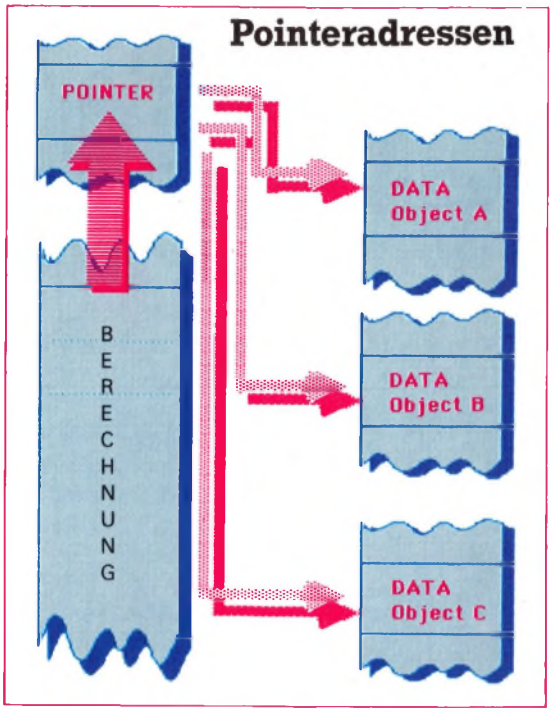
Adresse:	Code:	Label:	Befehl:	
1000	4278		CLR	COUNT
1002	3000			
-				
-				
-				
3000	0	COUNT	DS	1

Der Befehl CLR setzt dabei den Inhalt der Speicherstelle $3000 (COUNT) auf Null.

In der vorigen Folge hatten wir erwähnt, daß der PC von einem 32-Bit-Register dargestellt wird (von dem sich allerdings nur 24 Bits einsetzen lassen). Das bedeutet, daß die absolute Adresse, die den Operanden bezeichnet, nicht nur aus einem Wort – wie in den beiden letzten Beispielen – bestehen muß. Wie aber kann der Assembler feststellen, wieviel Platz er für die absolute Adresse reservieren soll? Da es Platzverschwendung wäre, wenn für jede absolute Adresse eine Langworterweiterung bereitgestellt würde, reserviert der Assembler – nachdem er die Operandenadresse festgestellt hat – eine Erweiterung in exakt der nötigen Länge.

Sie können den Assembler jedoch auch anweisen, für das ganze Programm lange oder kurze Erweiterungen je nach Bedarf zur Verfügung zu stellen.

Sehen wir uns einmal genauer an, wie in dem ADD-Beispiel die Langworterweiterung eingesetzt ist:



Adresse:	Code:	Label:	Befehl:
1000	D679	ADD	HIDATA,D3
1002	0020		
1004	0000		

Hier ist $200000 die absolute Adresse von HIDATA. Beachten Sie, daß der Code für den ADD-Bereich des Befehls (D6) gleich geblieben ist, die Operandenadresse jedoch von $78 auf $79 verändert wurde. Wir werden in späteren Artikeln über den 68000-Assembler den Einsatz von Langworterweiterungen noch ausführlich erklären.

Adressierungen

● **Registeradressierung:** Dies ist die einfachste Adressiermethode des 68000, da dabei nur die internen Register zum Einsatz kommen. So addiert ADD DO,D3 beispielsweise das Wort DO auf den Inhalt von D3.

Dieser Adressiermodus hat jedoch einige Einschränkungen. So können etwa Adreßregister nicht als Ziel eines Additionsvorganges angegeben werden – ADD DO,A4 ist nicht möglich. Mit dem Additionsbefehl ADDA läßt sich diese Einschränkung aber umgehen – ADDA DO,A4 ist legal.

Wenn Sie in den obigen Beispielen Langworte als Datenobjekte einsetzen wollen, brauchen Sie nur „.L" an den Befehl anzuhängen: ADD.L DO,D3 verarbeitet 32-Bit-Worte als Datenobjekte.

● **Indirekte Registeradressierung:** Diese Adressiermethode ist vermutlich die Wichtigste des 68000, da sie mit „Pointern" arbeitet. Auch lassen sich damit Stack-Vorgänge ausführen. Sehen wir uns zuerst die Pointer genauer an.

Programme müssen auf Datenobjekte – Bytes, Worte, Langworte oder strukturierte Objekte wie Arrays oder Records – „zeigen" können. Wenn beispielsweise ein Ablauf mehrfach für Daten des gleichen Typs ausgeführt werden soll, sind Pointer sehr praktisch. Das nebenstehende Bild zeigt, wie ein Pointer die einzelnen Elemente eines Records bezeichnet. Anfangs leitet er die Berechnung auf das Datenobjekt A, wird danach umgestellt und zeigt dann auf das nächste Datenobjekt.

Der Code des 68000 stellt den Pointer jedoch nicht direkt hinter den Befehl, sondern arbeitet mit einem Adreßregisterpointer. Dies mag in einigen Situationen etwas unbequem sein, doch stehen dafür acht Adreßregister zur Verfügung.

Hier ein Beispiel für eine einfache indirekte Adressierung:

```
LEA     INPUT,A0

MOVE.W (A0),D3
```

Der Befehl LEA lädt AO mit der Adresse eines Eingabedatenobjektes namens INPUT, das dann auf D3 kopiert wird (siehe Bild).
Wie läßt sich diese Adressiermethode für eine ganze Datenliste einsetzen? Zunächst brauchen wir eine „Nach-Inkrementierung". Hier wird der Pointer nach dem Zugriff auf das Datenobjekt inkrementiert und zeigt dann auf das nächste Element der Liste. Das nächste Beispiel zeigt, wie die Nach-Inkrementierung bei Computerworten arbeitet:

```
MOVE.W (A0)+,D3
```

Hier wird der Pointer nach dem Zugriff auf AO und dem Kopiervorgang auf D3 automatisch um Zwei inkrementiert. Die Adresse AO zeigt anfangs auf $2000, nach dem MOVE.W-Vorgang aber auf $2002. Für die Bearbeitung von Byteobjekten mit MOVE.B wird AO um Eins, bei MOVE.L um Vier inkrementiert.

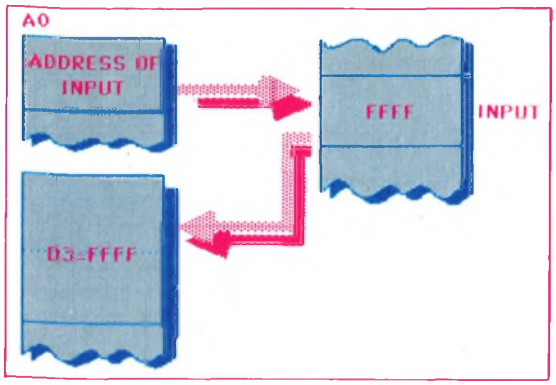
Wie flexibel diese Adressierungsart ist, zeigt sich deutlich in der folgenden Programmschleife zur Berechnung einer Liste:

```
LEA     INPUT,AO    Initialisieren
MOVE.W  (AO),D3     Jedes Element
                    kopieren
        :
        :
        Berechnung
        :
        :
        Auf Listenende
        prüfen
```

Bei jedem Ablauf der Berechnung deutet der Zeiger nach dem MOVE-Befehl automatisch auf das nächste Listenelement.

Es gibt jedoch auch die „Vor-Inkrementierung". Hier wird der Adreßpointer vor dem Zugriff auf das Datenobjekt dekrementiert.

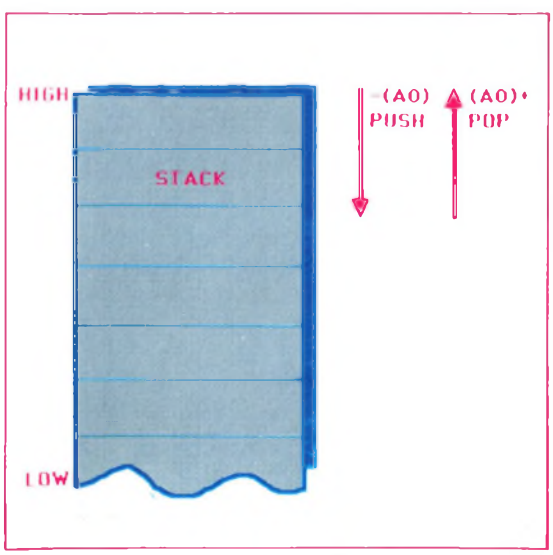
```
MOVE.W —(AO),D3
```



AO wird vor dem Kopieren des Datenobjektes auf D3 um zwei dekrementiert.

Die vor-inkrementierte indirekte Adressierung entspricht der indirekten Nach-Inkrementierung. Die Nach-Inkrementierung bearbeitet eine Liste in der Reihenfolge aufsteigender Adressen, während die Vor-Dekrementierung auf umgekehrte Weise vorgeht. Die vor- und nachgestellten Abläufe lassen sich jedoch nicht beliebig gegeneinander austauschen. So ist beispielsweise weder—(AO) noch (AO)+ möglich.

Sie mögen bereits bemerkt haben, daß die Adressierungsarten der Vor-Dekrementierung und des Nach-Inkrements den Stack-Vorgängen entspricht. Sie können damit Daten auf einen Stack „schieben" oder „herunterziehen", wobei ein Stack praktischerweise als eine Reihe von Adressen im Hi-Lo-Format verstanden wird. So legt MOVE DO,-(AO) Daten auf einen Stack, während MOVE (AO)+,DO sie wieder herunterzieht. Das Bild verdeutlicht diesen Vorgang des „Schiebens" und „Ziehens".



Die vertrauten Z80-Befehle PUSH und POP lassen sich auf dem 68000 mit den außerordentlich flexiblen Befehlserweiterungen der Vor-Dekrementierung und Nach-Inkrementierung simulieren.

Auf Stacks und ihren Einsatz gehen wir später in der 68000-Serie noch genauer ein, da diese praktischen Adressierarten leicht und gänzlich unkompliziert auf die entsprechenden Datenlisten zugreifen können.

Ereignisreich

In der letzten Folge hatten wir uns angesehen, wie das Schneider-Betriebssystem mit Interrupts arbeitet. Dabei hatten wir auch „Ereignisse" untersucht, die „Software Interrupts" darstellen. In diesem Artikel zeigen wir, wie sich Ereignisse in der Maschinencodeprogrammierung einsetzen lassen.

Das Schneider-Betriebssystem steuert Software-Ereignisse über einen „Ereignisblock". Dieser Block besteht aus sieben aufeinanderfolgenden Bytes, die sich mit der Routine KL_INIT_EVENT (bei $BCEF) an jeder beliebigen Stelle der mittleren 32K RAM anlegen lassen. Beim Aufruf der Routine muß das Registerpaar HL die Blockadresse enthalten, das Register B die Ereignisklasse (siehe Kasten), C die ROM-Wahl (0 bedeutet RAM) und DE die Adresse der Ereignisroutine. Unser Programmbeispiel zeigt diesen Ablauf.
Das Bild zeigt die Bitstruktur der im Register B gespeicherten Ereignisklasse. Der exakte Zeitpunkt, an dem ein Ereignis vom OS aufgerufen wird, hängt von Typ (synchron/asynchron) und der Priorität (normal/express) eines Ereignisses ab.

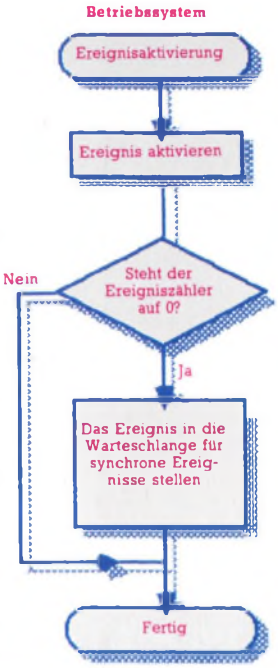
Das Betriebssystem behandelt synchrone und asynchrone Ereignisse unterschiedlich. Beide Ereignistypen lassen sich jedoch für jede Art von Ereignis verwenden.

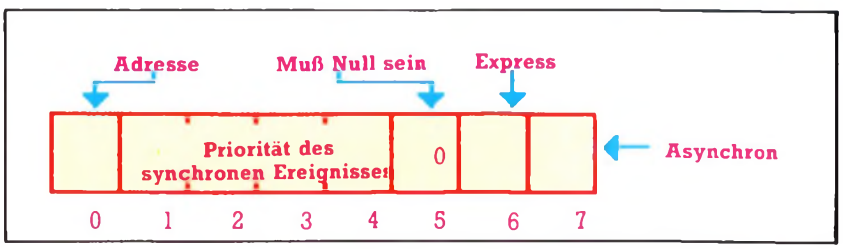
Der Ereigniszähler kann durch einen von vier Faktoren inkrementiert werden: vom Schnell-Interrupt, vom Ticker-Interrupt, vom Interrupt zur Bildauffrischung und von dem Jumpblock-Eintrag KL_EVENT. Den drei Timer-Interrupts sind Listen zugeordnet, in denen steht, welche Ereignisse beim Eintreten von Interrupts aktiviert werden. Die Routinen zur Anlage dieser Listen werden über die Jumpblock-Einträge aktiviert. Sie können entweder durch Initialisierung eines neuen Ereignisblocks Daten an die Liste anfügen oder einfach einen bereits bestehenden Block anhängen. Ereignisblöcke werden einzeln mit KL_INIT_EVENT initialisiert. Durch KL_EVENT lassen sich Ereignisse auslösen.

Asynchrone Ereignisse

„Normale" asynchrone Ereignisse werden zunächst in eine eigene Warteschlange gestellt. Darin sind alle asynchronen Ereignisse gespeichert, die während eines externen Interrupts ausgelöst wurden.

Nehmen wir als Beispiel einen normalen asynchronen Ereignisblock, der vom Ticker-Interrupt aktiviert wird. Beim Auftreten eines Interrupts stellt das Betriebssystem zunächst fest, welche Ereignisse überhaupt ausgelöst werden müssen (in diesem Fall unser Ereignis) und prüft, ob der entsprechende Ereigniszähler über Null steht. Danach wird jedoch das Ereignis nicht gleich ausgelöst, sondern erst in die Warteschlange gestellt. Nach Ablauf aller OS-Aufgaben, die der Ticker-Interrupt normalerweise auslöst, werden die Ereignisroutinen nacheinander aus der Warteschlange abgerufen. Da zuvor der Ticker-Interrupt wieder eingeschaltet wurde, werden auch alle später eintretenden Ereignisaktivierungen gespeichert. Die Routine kann daher beliebig lang sein – es gehen keine Ereignisse verloren. Schließlich löst der Aufruf einer Ereignisroutine auch die Dekrementierung ihres Zählers aus.

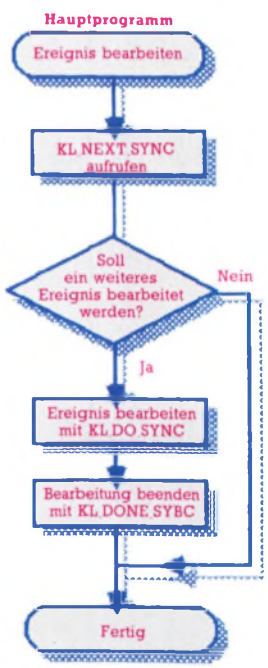




Asynchrone Ereignisse vom Typ „Express" (deren Zähler nach der Aktivierung über Null stehen) werden nicht erst in die Warteschlange gestellt, sondern sofort über ihre Ereignisroutinen aktiviert. Da dabei jedoch alle weiteren Interrupts abgestellt werden, können bei längeren Routinen externe Interruptanforderungen verlorengehen. Sie sollten diese Routine daher so kurz wie möglich halten – oder überhaupt nicht verwenden.

Bei synchronen Ereignissen entscheidet das Hauptprogramm den Zeitpunkt der Verarbeitung. Das Betriebssystem aktiviert daher das Ereignis nur und stellt es (bei einem Zähler über Null) je nach Priorität (siehe Ereignisblock) in die Warteschlange. Mit „Express" gekennzeichnete Ereignisse haben dabei Vorrang vor normalen Ereignistypen.

Zur Bearbeitung von Ereignissen führt ein Hauptprogramm bestimmte Schritte aus (siehe Flußdiagramm). Zunächst wird der Jumpblock-Eintrag KL_NEXT_SYNC aufgerufen, um festzustellen, welches Ereignis als nächstes in der Warteschlange steht. Das Ereignis wird dann über den Aufruf von KL_DO_SYNC ausgeführt. Dafür holt sich diese Routine die Adresse der



Die beiden Flußdiagramme zeigen die Reihenfolge der Abläufe, die das Betriebssystem (oben) und das Hauptprogramm (unten) bei der Bearbeitung von synchronen Ereignissen ausführen.

Ereignisroutine aus dem Ereignisblock und übergibt ihr die Steuerung. Nach Ablauf der Routine zeigt der Aufruf von KL_DONE_SYNC dem Betriebssystem an, daß die Bearbeitung beendet wurde. Wenn der Ereigniszähler nach der Dekrementierung über Null steht, wird der Ereignisblock wieder in die Liste der synchronen Ereignisse gestellt.

Jumpblöcke

In manchen Situationen muß es möglich sein, das Eintreten bestimmter Ereignisse zu verhindern. Hierfür bietet das Betriebssystem eine ganze Reihe von Jumpblock-Einträgen. So setzt KL_DISARM_EVENT den Zähler im Ereignisblock eines asynchronen Ereignisses auf einen Minuswert und verhindert so, daß diese Routine weiterhin aktiviert wird.

Zum Abschalten synchroner Ereignisse stehen drei Einträge zur Verfügung. KL_SYNC_RESET löscht alle fälligen Ereignisse aus der synchronen Warteschlange, ändert aber deren Zähler nicht. Das Ereignis ist abgeschaltet, da der Ereigniszähler nur dekrementiert werden kann, wenn sich das Ereignis in der Warteschlange befindet. KL_DEL_SYNCHRONOUS löscht alle Ereignisse einer Warteschlange. KL_EVENT_DISABLE verhindert, daß normale synchrone Ereignisse in die Warteschlange gestellt werden, läßt die Aktivierung von Ereignissen aber weiterhin zu.

Obwohl Ereignisse auf den ersten Blick sehr kompliziert erscheinen, können Sie den Programmierer vor Problemen schützen, die bei Interrupts auftreten. Sie müssen jedoch genau darauf achten, daß die Daten der asynchronen Ereignisse vom Typ „Express" nicht mit dem Hauptprogramm kollidieren, sonst kann es zu folgenschweren Fehlern kommen.

Hintergrund-Piepser

Unser Programmbeispiel zeigt, wie mit den im Artikel beschriebenen Techniken Ereignisse angewendet werden. Dabei wird ein Ticker-Ereignis als Sekundenzähler angelegt, der wiederum veranlaßt, daß der Computer alle Sekunde ein „Beep" hervorbringt. Der Aufruf des Programms über seine Assembleradresse initialisiert die Ereignisse.

Das Piepen setzt sich auch bei einem laufenden BASIC-Programm im Hintergrund fort. Wenn das BASIC-Programm ebenfalls den Tongenerator anspricht, können seltsame Wirkungen entstehen.

```
; This program demonstrates the use of several
; types of event to generate a simple clock that
; sounds a beeper every second.
;
INIT.EVENT:     EQU  £BCEF ; KL_INIT_EVENT
ADD.TICK:       EQU  £BCE9 ; KL_ADD_TICKER
KICK:           EQU  £BCF2 ; KL_EVENT
TXT.OUT         EQU  £BB5A ; TXT_OUTPUT
;
BLEEP:           EQU  £07
;
; First initialise the event blocks
;
                ORG  £8000
;
                LD   HL, TICK  ; start with ticker
                LD   B, £81    ; async, express
                LD   C, 0      ; no rom select
                LD   DE, FRAC  ; routine address
;
                CALL INIT.EVENT
;
; BC is preserved
;
                LD   HL, SEC   ; seconds counter
                LD   DE, SECND ; routine address
                CALL INIT.EVENT
;
                LD   HL, BP    ; bleeper routine
                LD   DE, BEEP  ; routine address
                CALL INIT.EVENT
;
; now log on the Ticker Routine
;
                LD   HL, FRBLK ; tick block address
                LD   DE, 1     ; count
                LD   BC, 1     ; recharge
                CALL ADD.TICK  ; log it in
;
                RET
;
;Event Processing Routines
;
FRAC:
; maintains a 1/50th of a second count
;
                LD   HL, SEC50 ; point to count
                LD   B, 50     ; one second?
                CALL TEST
                RET  NZ        ; no, then finish
                LD   HL, SEC   ; kick next second
                CALL KICK
                RET
;
SECND:
; This is called once a second - kicks a beep
                LD  HL, BP     ; kick beep event
                CALL KICK
                RET
;
BEEP:
; does a beep
                LD   A, BLEEP  ; send a bleep char
                CALL TXT.OUT
                RET
;
TEST:
; tests a counter
; B contains the required value on entry
; HL points to the count store location
;
                INC  (HL)      ; update it
                LD   A, (HL)   ; read the count
                CP   B         ; finished
;
                RET  NZ        ; no, return
                XOR  A         ; set zero true
                LD   (HL), A   ; reset the count
                RET            ; and go back
;
; now the event and tick blocks
;
FRBLK:          DEFS 6         ; tick block space
;
TICK:           DEFS 7         ; ticker event block
;
SEC:            DEFS 7         ; 1 sec event block
;
BP:             DEFS 7         ; beep event block
;
SEC50:          DEFB 0         ; 1/50 sec counter
```

Geänderte Adressierungen

In der vorigen Folge hatten wir uns die ersten drei Register-Adressierungsarten angesehen. In diesem Artikel beschäftigen wir uns mit den übrigen vier Adressiermethoden und ihren Unterschieden.

Sehen wir uns kurz die bereits beschriebenen Adressierungsarten an:

Absolute Adressierung: Der Quell- oder Zieloperand ist eine Speicheradresse.

- **Register-Adressierung:** Quelle oder Ziel ist ein Register.
- **Indirekte Register-Adressierung (mit Pointer):** Der Pointer zeigt auf das Quell- oder Zielobjekt.
- **Indirekte Register-Adressierung (mit Nach-Inkrementierung oder Vor-Dekrementierung):** Eine Datenliste wird von oben nach unten oder umgekehrt durchgearbeitet.

Bei der „unmittelbaren Adressierung" ist eine Konstante direkt hinter dem Befehl untergebracht, so daß die Anweisung oft nur ein Computerwort belegt.

- **Unmittelbare Adressierung:** Hier ist der Quelloperand eine Konstante. MOVE.W #$43,D0 speichert die mit # gekennzeichnete Konstante $43 in D0 ($ bedeutet Hexadezimalformat). Diese Art der Adressierung heißt unmittelbar, da das konstante Computerwort direkt hinter dem Befehl steht. Bei Operanden mit Byte-Format belegt die Erweiterung natürlich nur ein Byte, bei Langworten vier Bytes.

Der unmittelbare Modus läßt sich gut für Konstantendeklarationen einsetzen, beispielsweise für eine Schleife mit vier Wiederholungen. Es gibt jedoch Situationen, in denen Sie eine Konstante besser in eine mit Namen gekennzeichnete Adresse laden, die dann absolut adressiert wird. Hier ein Beispiel:

```
MOVE    COUNT,D0
MOVE    COUNT,D5
COUNT   DC.W 4
```

Diese Methode ist flexibler, wenn der Wert von COUNT einmal geändert werden muß. Bei der absoluten Adressierung brauchen Sie nur den Inhalt von COUNT zu ändern, während bei der unmittelbaren Adressierung dafür erst mühsam alle Vorkommen der Konstanten gesucht werden müssen. Diese Suche ist weiterhin sehr fehleranfällig.

Auch bestimmte „Quick"-Befehle erlauben den unmittelbaren Modus, wobei Opcode und Konstante dann in einem Computerwort stehen. Die folgenden zwei Befehle

```
ADDQ    #3,D0
SUBQ    #1,D3
```

sind in einem Wort codiert und werden daher auch schneller ausgeführt. Beachten Sie dabei, daß die Konstanten bei ADDQ und SUBQ nur zwischen #1 und #8 liegen dürfen.

Auch der Befehl MOVE hat eine „Quick"-Version. Hier sind jedoch Konstanten im Bereich von −128 bis +128 erlaubt. MOVEQ #98,D4 speichert 98 (dezimal) in D4.

Die folgende Tabelle erhält ein Beispiel von jeder bisher behandelten Adressierungsart:

1	LEA	OUTPUT,A1	Absolut und Register
2	MOVE	A1,A2	Register
3	MOVE	-(A1),D3	Vor-Dekrementierung und Register
4	ADDQ	#3,D3	Quick Unmittelbar und Register
5	MOVE	D3,(A2)+	Register und Nach-Inkrementierung
6	ADD	#6,D3	Unmittelbar und Register
7	MOVE	D3,(A2)	Register und Indirekt
8	INPUT	DC.W	#6 Konstante 6
9	OUTPUT	DS.W	2Platz für zwei Worte

Die Speicherstelle INPUT enthält die Konstante 6, die Adresse OUTPUT hat zwei Wörter zur Verfügung.

Wie sieht nun der Inhalt von OUTPUT und OUTPUT+2 nach der Ausführung dieses Codes aus? Zeile 1 und 2 setzen A1 und A2, die dann auf OUTPUT zeigen. Zeile 3 läßt A1 auf INPUT zeigen, bevor der Operand adressiert wird. Danach wird 6 in D3 geladen. Zeile 4 addiert 3 auf D3, das dann 9 enthält.

Zeile 5 lädt D3 in OUTPUT, da A2 auf OUTPUT zeigt. Nun wird A2 um Zwei inkrementiert und zeigt so auf das zweite Wort von OUTPUT. Zeile 6 addiert 6 auf den Inhalt von D3 (D3 enthält damit den Wert 15), der dann in das zweite Wort von OUTPUT geladen wird.

- **Indirekte Adressierung (mit Adreßdistanzwert und Index):** Zunächst wollen wir untersuchen, was „Adreßdistanzwert" und „Index" überhaupt bedeuten. Auf dem 68000 bezieht sich der Adreßdistanzwert auf einen festen

Das Anwenderhandbuch für den 68000 enthält kurze Beschreibungen der Befehle und Einsatzarten. Eine Reihe von Beispielprogrammen zeigen, wie sich die Möglichkeiten des 68000 optimal einsetzen lassen. Herausgeber: Sigma Press ISBN 1-85058-001-4.

Offset (oder Distanz) von einem Basiswert aus, während Index per Register einen variablen Distanzwert angibt.

Das nebenstehende Bild zeigt den Unterschied zwischen diesen beiden Begriffen:
1) das Adreßregister „An" zeigt auf ein strukturiertes Datenobjekt namens Z;
2) die interne Substruktur Y wird über das Indexregister „Ri" angesprochen; während schließlich
3) auf das Datenelement X (der Struktur Y) mit dem festen Distanzwert zugegriffen wird.

Beispiele für den Einsatz sind:
1) Z: ein Array von Datensätzen oder ein zweidimensionales Array;
2) Y: ein einzelner Datensatz oder eine Arrayzeile;
3) X: das Daten- oder Arrayelement (Ganzzahl).

Da strukturierte Datenelemente dieser Art in modernen Hochsprachen wie PASCAL oder ADA häufig vorkommen, haben diese Zugriffsmethoden große Bedeutung. Natürlich gibt es keinen Grund, Assemblerprogramme nicht ebenfalls auf diese Weise zu strukturieren; der 68000 jedenfalls bietet diese Möglichkeit.

Doch zurück zu unserem strukturierten Datenobjekt. Sie haben sicherlich bemerkt, daß die Adresse von X aus der Summe von An+Ri+Adreßdistanzwert gebildet wird und An und Ri bei der Programmausführung geändert werden können. Sehen wir uns dazu einige konkrete Beispiele an. Die einfachste Methode ist die indirekte Adressierung mit Distanzwert:

```
MOVE.W DISP(A0),D1
```

DISP wurde zuvor als symbolischer Name definiert und mit einem Wert, zum Beispiel 6, belegt. Der Befehl adressiert daher die Quelle indirekt mit dem Distanzwert 6. Wenn beispielsweise A0 auf die Adresse $1000 zeigt, dann wird D1 mit dem Inhalt von $1006 geladen. Da der Befehl Wortformat hat, ist ein 16-Bit-Adreßdistanzwert möglich (von +32767 bis −32768).

Das nächste Beispiel zeigt die flexibelste Adressierungsart, die der 68000 bietet:

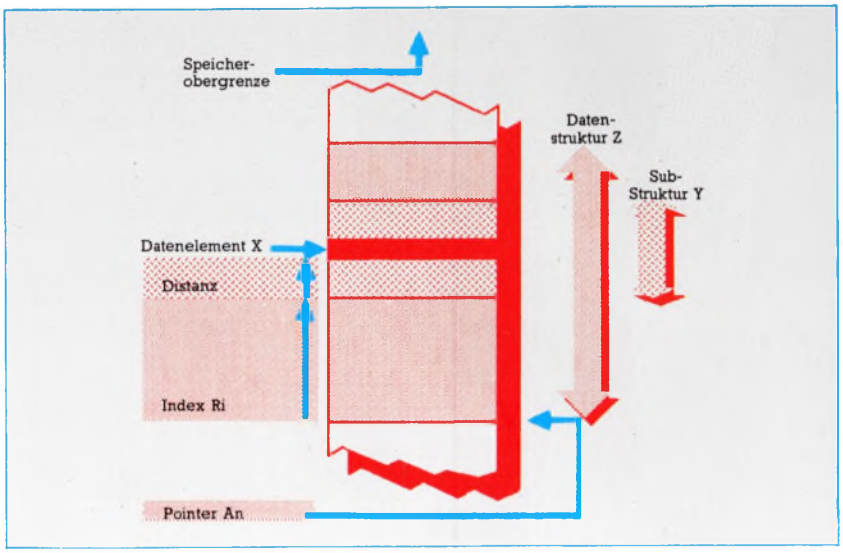
```
MOVE.W DISP(A0,D0.W),D1
```

Die Quelladresse wird durch Addieren des Basisregisters A0, des Indexregisters D0 und des festen Distanzwertes DISP bestimmt. DISP kann dabei jedoch nur eine vorzeichenbehaftete Ganzzahl im Acht-Bit-Format sein, während im Indexregister das volle 32-Bit-Format zur Verfügung steht.

Bei diesen beiden Adressierungsarten (indirekt und Distanzwert, mit oder ohne Index) wird der Distanzwert bei der Assemblierung festgelegt. Wenn Sie einen Distanzwert dynamisch ändern wollen, steht Ihnen

```
MOVE.W 0(A0,D1),D2
```

zur Verfügung, wobei D1 die Rolle des Distanzwertes übernimmt und der feste Distanzwert auf Null steht.



Der Befehlssatz des 68000 bietet diese flexiblen Adressierungsarten für die meisten der oft eingesetzten Befehle. Nachfolgend einige Adressierbeispiele mit Distanzwerten und Indizes (im Datenformat Byte):

```
1   LEA     LIST,A1
2   MOVEQ   #4,D1          D1:=4
3   MOVE.B  2(A1),D6       D6:=3
4   ADD.B   0(A1,D1),D6    D6:=8
5   MOVE.B  D6,4(A1,D1)    LIST+8:=8
6
7   LIST    DC.BQ          1,2,3,4,5,6,7,8,0
```

Zeile 1 veranlaßt, daß A1 auf LIST zeigt. Danach wird D1 auf 4 gesetzt. Zeile 3 addiert den Distanzwert 2 auf den Pointer in A1 und kann so das dritte Element von LIST in D6 laden. Danach wird mit dem Index 4 (in D1) der Inhalt von Element 5 auf D6 addiert und das Ergebnis in LIST+8 gespeichert.

● **Adressierung relativ zum Programmzähler (PC):** Bevor wir uns mit PC-relativen Adressiermethoden beschäftigen, sehen wir uns einige der Assembleranweisungen an. Assembleranweisungen erzeugen keinen ausführbaren Code, beeinflussen aber Faktoren wie Listenausgabe und Symboldefinitionen. Die Anweisung ORG (englisch „Origin" — Ursprung) bezieht sich auf die Adresse des Programmanfangs und die Art des erzeugten Codes.

Normalerweise wird eine Anfangsadresse mit

```
ORG $1000
```

angegeben. Damit ist die Startadresse (oder besser: die Ladeadresse für den Binärlader) auf $1000 gesetzt, so daß der darauffolgende Code in eine sequentiell aufsteigende Adresse geladen wird. Das Auftreten einer weiteren ORG-Anweisung setzt von diesem Punkt an natürlich eine neue Ladeadresse. Eine Variante ist

Das Buch „68000 Assembly Language Programming", herausgegeben von Osborne/McGraw-Hill, enthält eine umfassende Einführung in die Programmierung des 68000. Erfahrene Programmierer werden die Teile über die Theorie der Assemblerprogrammierung überflüssig finden. Das Buch enthält jedoch alle wichtigen Informationen. Sehr praktisch sind die fettgedruckten Absatzüberschriften, durch die sich einzelne Informationen leicht finden lassen. ISBN 0-931988-62-4.

```
ORG.L $2000
```

bei der alle Adreßbezüge als Langworte angelegt sind und absolute Adressen jeweils zwei Computerwörter belegen.

Die Anweisung RORG definiert eine Ladeadresse, von der aus alle Speicheradressen relativ zum Programmzähler angelegt sind. Diese Art Code läßt sich in jeden beliebigen Speicherbereich laden, da sich alle Adressen auf die Ladeadresse beziehen. Hier ein Bei-

```
                    RORG    $2000
2000 D47A START     ADD     CONST1,D2
2002 000A
2004 3202           MOVE    D2,D1
2006 6000           BRA     START
2008 FFFB
200A 4E40           TRAP    #0
200C 0026 CONST1    DC.W    38
```

Zur Adreßberechnung von CONST1 wird hier der 16-Bit-Distanzwert $A (dezimal 10) in Speicherstelle $2002 auf den PC addiert. Da der PC beim Laden des Distanzwertes auf $2002 steht, ist $200C die Adresse der Quelldaten. Es gibt keine speziellen Befehle oder Adressierungsarten, die PC-relativen Code erzeugen, die Anweisung RORG reicht aus.

Der BRA-Befehl

Noch einige weitere Informationen über diesen Code. Der dem BRA-Befehl (unbedingte Verzweigung) zugeordnete Distanzwert befindet sich in $2008. Durch Addieren des PC ergibt sich die Operandenadresse von START. Das bedeutet, daß der BRA-Befehl immer PC-relativen Code erzeugt. Weiterhin wird der Befehl TRAP als Stopmarkierung eingesetzt. Bei Verwendung eines Monitors stellt er die Register dar und vereinfacht damit wesentlich die Fehlersuche.

Hier ein Beispiel, in dem die PC-relative Adressierung einen negativen Distanzwert ergibt:

```
                    RORG    $2000
2000 0026 CONST1    DC.W    38
2002 D47A START     ADD     CONST1,D2
2004 FFFC
2006 3202           MOVE    D2,D1
2008 6000           BRA     START
200A FFF8
200C 4E40           TRAP    #0
```

Der Bezug auf CONST1 in Adresse $2004 erhält einen negativen Distanzwert.

Diese Art der Adressierung läßt sich jedoch nur für den Quelloperanden einsetzen, bei

```
                    RORG    $2000
TOTAL               DC.W    0
START               ADD     D2,TOTAL
```

wird ein Assemblerfehler angezeigt. Diese Art der Adressierung eignet sich speziell für Code, der für feste ROM-Adressen bestimmt ist, zuvor aber im RAM getestet werden soll.

Es mag Ihnen wie eine Einschränkung vorkommen, daß dieser Adressiermodus nicht in Speicherstellen schreiben kann, doch steht Ihnen der „Bezug" auf absolute Adressen jederzeit zur Verfügung:

```
                    RORG    $1000
TOTAL               DC.W    0
                    RORG    $2000
CONST1              DC.W    38
START               ADD     CONST1,D2
                    MOVE    D2, TOTAL
```

Hier ist der Bezug auf TOTAL legal, da ein absoluter Adreßbereich angesprochen wird.

Die PC-relative Adressierung kann auch um einen Index mit Distanzwert erweitert werden. Hier ein Beispiel:

```
                    RORG    $3000
INDEX               DC.W    10
START               MOVEA   INDEX,A5
                    ADD     6(A5),D2
```

In diesem Fall müssen wir den Befehl MOVEA einsetzen, um A5 mit dem Inhalt der Speicherstelle INDEX laden zu können. LEA läßt sich nicht verwenden, da damit die Adresse von INDEX angesprochen würde und bei der PC-relativen Adressierung auch nicht erlaubt ist. Auch ein MOVE-Befehl wäre illegal, da ein Adreßregister nicht Zieloperand sein kann.

Doch zurück zu unserem Beispiel, in dem sich der Quelloperand des ADD-Befehls nach der Ausführung aus dem PC-Wert plus Indexregister (A5) plus Distanzwert (6) errechnet. Dabei liegt die Quellenadresse 16 Bytes hinter der Adresse mit dem Erweiterungswort 6.

Diese Adressierungsart wird allerdings dadurch eingeschränkt, daß der Distanzwert nur acht Bits des Opcodes belegen kann und damit auf Werte zwischen −128 und +127 beschränkt ist.

Beliebige Ausführung

PC-relativer Code läßt sich an jeder beliebigen Stelle des Speichers ausführen. Die Adressiermethode wird daher hauptsächlich für die Entwicklung positionsunabhängiger Codes eingesetzt. Sie ist besonders wichtig bei Programmen mit vielen Modulen, in denen die Ladeposition eines Moduls erst beim Aufruf feststeht. Für große vielschichtige Programmiersysteme sind natürlich spezielle Speicherverwaltungssysteme nötig, doch für alle einfacheren Programmstrukturen hat die PC-relative Adressierung große Bedeutung.

- **Implizierte Adressierung:** Diese Methode ist unkompliziert, da die eingesetzten Register schon im Opcode enthalten sind. So beeinflußt beispielsweise RTS den PC und den Stackpointer, und BRA spricht den PC an.

Zu Befehl!

In den nächsten Folgen dieser Serie gehen wir ausführlich auf den Befehlssatz des Motorola 68000 ein. Zuerst untersuchen wir Übertragungsbefehle wie MOVE und PEA und sehen uns die arithmetischen Anweisungen an.

In den vorangegangenen beiden Folgen hatten wir gesehen, wie der Motorola 68000 Operanden adressiert, und dabei einen Einblick in Grundbefehle wie MOVE und ADD erhalten. Wir werden nun zunächst die Datenübertragungsbefehle untersuchen und uns dann den Arithmetikbefehlen für binäre Berechnungen zuwenden.

Wir beschreiben dabei die Anwendung der wichtigeren Befehle und gehen auch auf Hindernisse und wesentliche Einzelheiten ein. Wenn Sie viel mit dem 68000 arbeiten wollen, sollten Sie sich das Buch „Programmierung des 68000" oder ein anderes der in dieser Serie empfohlenen Handbücher zulegen.

Bevor wir die Befehle im einzelnen behandeln, sehen wir uns erst einmal den Inhalt des Statusregisters (SR) genauer an. Die eine Hälfte dieses Registers enthält die Bedingungscodes mit Informationen über das Ergebnis des letzten ausgeführten Befehls. Jeder Bedingungscode läßt sich als ein Ein-Bit-Speicher ansehen, der einem bestimmten arithmetischen Zustand zugeordnet ist. Hier die Codes im einzelnen:

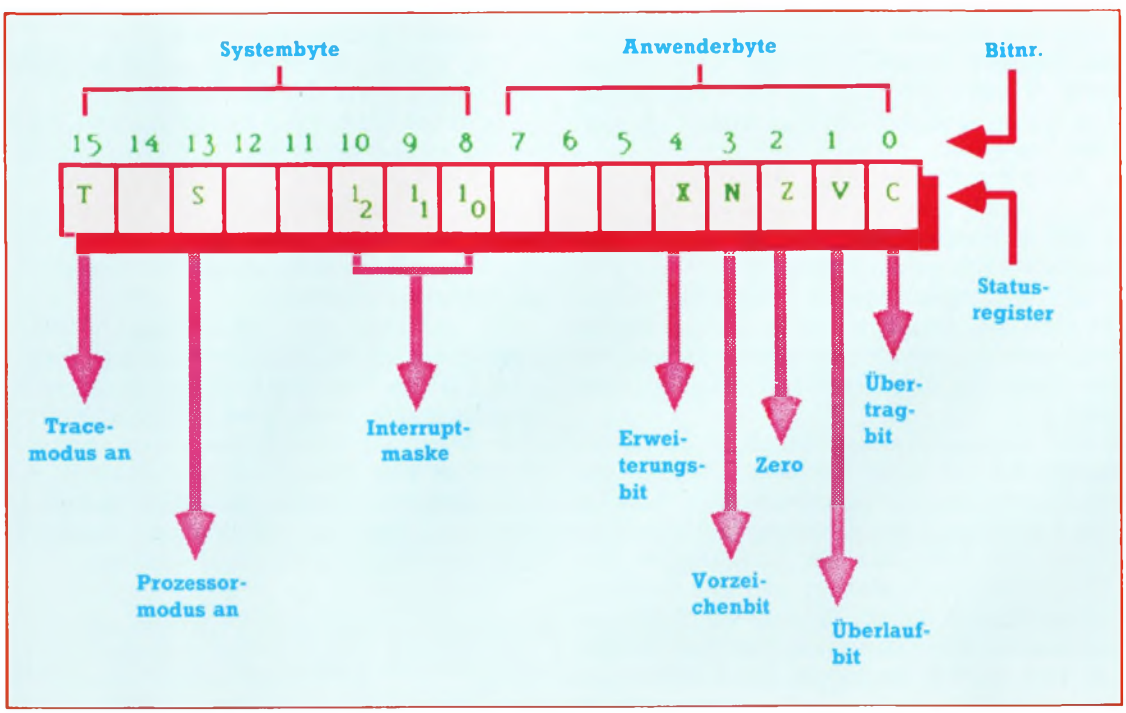
- **BIT 0: Übertrags-Bit oder C-Bit:** Dieses Bit wird gesetzt, wenn ein arithmetischer Vorgang einen Übertrag aus dem höchstwertigen Bit des Datenoperanden ergibt, zum Beispiel:

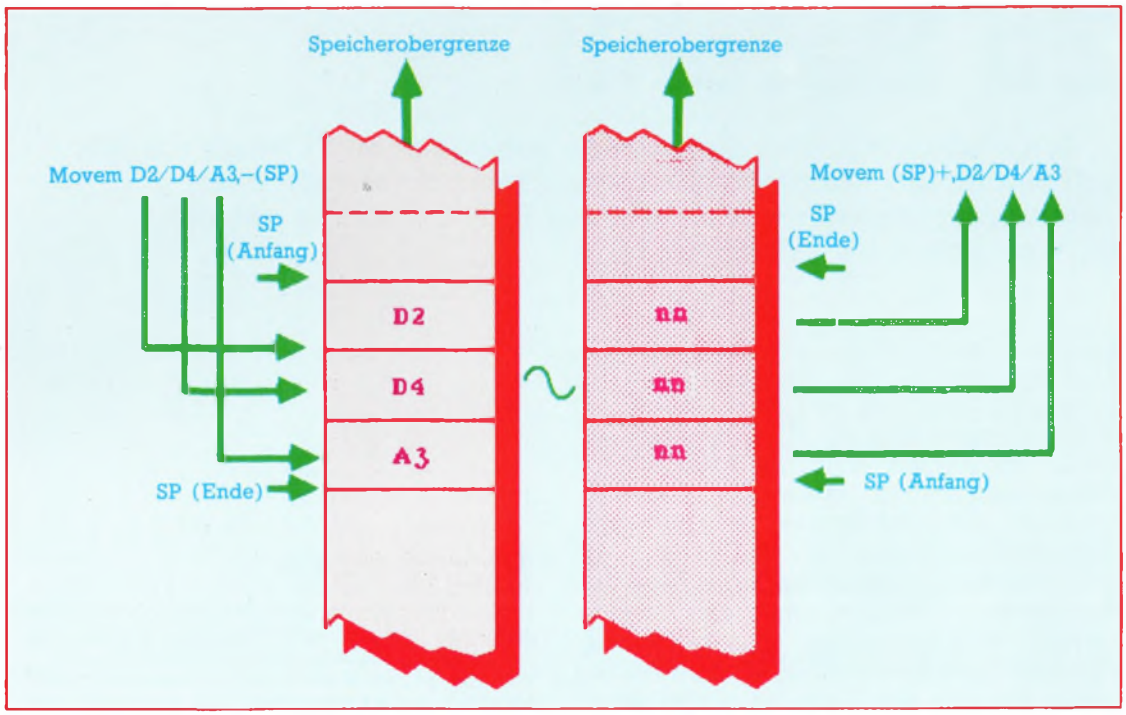
```
Die Addition 01100000
von          11100000
             ---------
ergibt       101000000
```

Statussymbole

Das Statusregister im 32-Bit-Format ist in einen System- und einen Anwenderbereich unterteilt. Unser Bild zeigt die Bedeutung der einzelnen Bits. Ein Statusbit gibt an, in welchem Prozessormodus der 68000 arbeitet, auf Supervisor- und Anwenderebene. Die Überwachung der verschiedenen Arbeitsabläufe ist im Mehrplatzbetrieb notwendig. Normalerweise werden auf dem 68000 alle Aufgaben im Anwendermodus (der über einen eigenen Stackpointer verfügt, aber die Statusbits nicht verändern kann) ausgeführt. Die Systemsoftware (beispielsweise die Sytemsteuerung) läuft im Supervisormodus ab.

Am Anfang werden Sie die Arbeit im Supervisormodus sicherlich einfacher finden, da darin alle Opcodes legal sind, während die Programmierung im Anwendermodus Einschränkungen unterworfen ist. Der 68000 verfügt weiterhin über die Betriebsart ‚Trace', mit der Sie zur Fehlersuche Programme schrittweise ablaufen lassen können. Dabei ruft der 68000 nach jedem Befehl automatisch eine Korrekturroutine auf.





Mit MOVEM können Sie in einem Befehl ganze Registerlisten übertragen. Das Bild zeigt, wie MOVEM mit der Vor-Dekrementierung und der Nach-Inkrementierung drei Register auf den Stack schiebt und wieder herunterzieht. Dabei wird auch die Position des Stackpointers vor und nach jedem Befehl angegeben.

Hier wird die 1 aus dem höchstwertigen Bit des Ergebnisses (mit Byte-Länge) übertragen.

Der Übertrag fließt jedoch nicht in das niederwertige Bit der nächsthöheren Struktur (in diesem Fall – des Computerwortes), sondern in das C-Bit des SR. Ob der Übertrag überhaupt Bedeutung hat, hängt allerdings direkt vom programmierten Ablauf ab. Bei Berechnungen mit höherer Präzision (z. B. wenn der Datenoperand aus mehreren Einheiten besteht) hat der Übertrag große Bedeutung.

- **BIT 1: Überlauf-Bit oder V-Bit:** Dieses Bit wird gesetzt, wenn das Ergebnis der Berechnung nicht in den Datenoperanden paßt. So erzeugt das Addieren von 1 auf 32767 (die höchste positive Ganzzahl bei einem 16-Bit-Computerwort) einen Überlauf im Datenoperanden. Das binäre Ergebnis läßt sich damit nicht korrekt darstellen.
- **BIT 2: Null-Bit oder Z-Bit:** Wird gesetzt, wenn das Ergebnis einer Berechnung Null ist.
- **BIT 3: Vorzeichen-Bit oder N-Bit:** Wird bei negativen Ergebnissen gesetzt.
- **BIT 4: Erweiterungs-Bit oder X-Bit:** Dieses Bit wird bei Vorgängen mit höherer Präzision eingesetzt. Es hat die gleichen Funktionen wie das C-Bit, wird aber von MOVE-Befehlen nicht verändert.

Der flexible Übertragunsbefehl MOVE kopiert Daten von einer Quelle zum Ziel. Er kann die Quelle beliebig adressieren und auch für das Ziel die meisten Adressiermethoden einsetzen (mit Ausnahme der Adreßregister, der PC-relativen Adressierung und des unmittelbaren Modus). Diese Gruppe von Adressierungsarten wird „datenverändernder Modus" genannt. Eine Untergruppe davon stellen die sogenannten „speicherverändernden Modi" dar, die alle Daten mit Ausnahme der Datenregister verändern können.

Beim MOVE-Befehl sind folgende Anweisungen illegal:

```
RORG $1000
MOVE D2,BETTY   PC-relativ ist für BETTY
                nicht möglich
MOVE D2,A2      Adreßregister lassen sich
                nicht als Ziel einsetzen
```

MOVE beeinflußt Bit N und Z des SR, Bit V und Z werden immer auf Null gesetzt.

Adreßregister können auf zwei Wegen als Zieloperand eingesetzt werden:

- MOVEA kopiert den Inhalt des Quelloperanden in das Adreßregister (Ziel).
- LEA kopiert die (normalerweise absolute) Quellenadresse in das Adreßregister.

Diese beiden Befehle haben keinen Einfluß auf die Bedingungscodes. Nach dem gleichen Prinzip arbeiten auch Spezialbefehle, die Daten ins Statusregister und den Stackpointer setzen oder von dort holen. Diese Befehle werden jedoch hauptsächlich in der Systemprogrammierung eingesetzt.

Ein weiterer, sehr brauchbarer Übertragungsbefehl ist MOVEM, der beliebige Daten- oder Adreßregister aus aufeinanderfolgenden Speicherstellen lesen oder dorthin schreiben kann. Damit lassen sich beispielsweise beim Eintritt in eine Subroutine alle Register, die dort verändert werden, mit einem Befehl sichern und nach dem Rücksprung wieder in ihren ursprünglichen Zustand versetzen:

```
Einsprung MOVEM   D2/D4/A3,PAD
                  (die Subroutine
                  arbeitet mit D2,D4
                  und A3)
Rücksprung MOVEM PAD,D2/D4/A3
```

Beim Sprung auf die Subroutine werden D2,D4 und A3 in PAD gesichert und nach Beendigung wieder zurückgeladen. Sie können die Register aber auch mit dem Stack sichern:

```
Einsprung MOVEM   D2/D4/A3,-(SP)
                  (Code der Subroutine)
Rücksprung MOVEM (SP)+,D2/D4/A3
```

Eine Variation des Befehls MOVE ist die Schnellübertragung (MOVEQ). Da der ganze Befehl in einem Computerwort untergebracht ist, eignet er sich besonders für das Setzen einer vorzeichenbehafteten Acht-Bit-Konstanten (+127 bis −128). MOVEQ initialisiert oft Datenregister als Schleifenzähler. Das Wort

```
MOVEQ #34,D2
```

überträgt die Zahl 34 in D2. Wenn das Q fehlt, nehmen einige Assembler nicht automatisch den schnellen Modus, sondern codieren MOVE #34,D2 in zwei Wörtern.

PEA (oder „effektive Adresse auf den Stack schieben") ist ein weiterer Datenübertragungsbefehl, der sich gut für Stackvorgänge eignet. So schiebt PEA BETTY die Adresse BETTY auf den Stapel und vermindert den Stackpointer.

Schließlich gibt es noch Befehle zum Austauschen von Registern. SWAP vertauscht die 16 unteren Bits eines Registers mit den 16 oberen Bits.

```
SWAP D2
```

tauscht das Computerwort in Bit 0 bis Bit 15 gegen das Wort von Bit 16 bis Bit 31. Der Befehl läßt sich gut für Berechnungen einsetzen, die in den Datenregistern mit Langwörtern arbeiten. Die Prozedur sieht so aus:

- Langwort in D2 laden
- Berechnung mit dem Wort ausführen
- SWAP D2
- Berechnung mit dem Wort ausführen
- SWAP Wort

Mit dem Tauschbefehl EXG lassen sich in bestimmten Kombinationen Langworte im 32-Bit-Format tauschen. Hier einige Beispiele:

```
EXG D2,D3 tauscht Datenregister
EXG A3,A4 tauscht Adreßregister
EXG D2,A5 tauscht Adreß- und Datenregister
```

EXG ist eigentlich ein SWAP mit 32-Bit-Worten.

Diese Befehle bilden die Grundlage aller mathematischen Berechnungen. Sie addieren Binärstellen auf Basisebene. Selbst wenn es in Ihrem Programm keine numerischen Vorgänge gibt, finden einfache arithmetische Vorgänge statt, wenn Zeichencodes umgewandelt oder Arrayindizes angelegt werden.

Der Befehl ADD addiert die Quelle mit dem Ziel und speichert das Ergebnis im Ziel. Das Datenobjekt kann jede mögliche Größe annehmen, es werden alle Bedingungscodes beeinflußt. So addiert ADD.W D2,D3 den Inhalt von D2 mit D3 und speichert das Ergebnis in D3.

Für den Quelloperanden können alle Adressierarten eingesetzt werden. Wenn das Ziel ein Adreßregister sein soll, müssen Sie allerdings mit ADDA arbeiten. ADDA D2, A4 addiert daher den Inhalt von D2 auf A4.

Für unmittelbare Daten gibt es den Spezialbefehl ADDI, der diese Informationen (alle Datenformate sind möglich) als Erweiterungswort im Ziel speichert (hier sind nur datenverändernde Modi erlaubt). ADDI #3423 BETTY addiert 3423 mit dem Inhalt von BETTY.

Wie bei MOVE gibt es auch eine schnelle Version von ADDI — ADDQ. Die Daten dürfen jedoch nur im Bereich von 1 bis 8 liegen. ADDQ 5,D2 addiert 5 auf den Inhalt von D2, wobei der ganze Befehl nur ein Wort belegt.

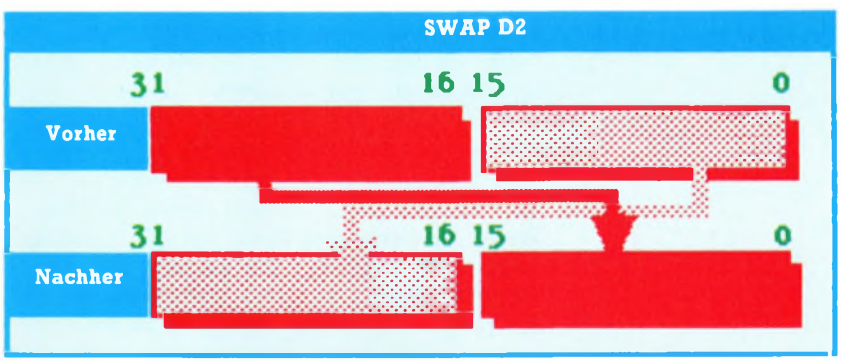
Alle bisher erwähnten Addierbefehle berücksichtigen keine Überträge. Für die Bearbeitung von Überträgen, besonders bei Berechnungen mit mehrfacher Präzision, müssen Sie den Befehl ADDX verwenden. ADDX D2,D4 addiert den Inhalt von D2, D4 und dem Erweiterungsbit in SR und speichert das Ergebnis in D4. Wenn D2 00000000 enthält, D4 00000001 und X auf 1 steht, dann ist der Inhalt von D4 nach Ausführung von ADDX 00000010. Mit ADD erhalten Sie in D4 den Wert 00000001.

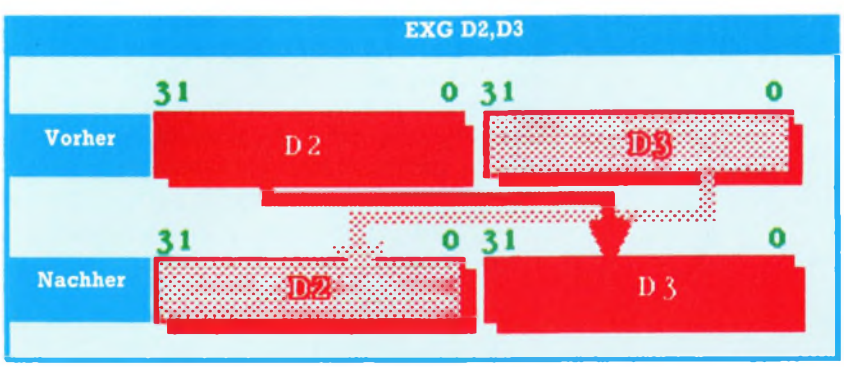
Die SUBtraktionsbefehle entsprechen den ADD-Anweisungen. Sie haben die gleichen Adreßbeschränkungen und setzen auch die gleichen Bits des SR. Die folgenden Beispiele sind alle legal:

```
SUB D2,D3    D2 von D3 subtrahieren und
             das Ergebnis in D3 speichern
SUBA #4,A3   4 von A3 subtrahieren
SUBI #200,D2 200 von D2 subtrahieren
SUBQ #1,D2   1 von D2 schnell subtrahieren
SUBX D2,D4   Speichert das Ergebnis von
             D4–D2–X in D4
```

Beachten Sie, daß der 68000er keinen separaten In- oder Dekrementierbefehl bietet. Sie müssen dafür die Versionen von ADD und SUB einsetzen, die nur ein Wort lang sind.

Die Arbeit mit Langwörtern kann recht unhandlich werden, wenn Sie mit Wort- oder Byteoperanden arbeiten. SWAP erleichtert diese Aufgabe, indem es die Registerwerte von Bit 0 bis Bit 15 gegen die von Bit 16 bis Bit 31 austauscht. Mit EXG lassen sich Langwörter gegeneinander austauschen.





Bildschirmtricks

Das Schneider-Betriebssystem unterstützt eine Textanzeige, die völlig von der Grafikdarstellung getrennt ist. Wir untersuchen, welche Bildschirmsteuerungen interessant sind und zeigen drei Module, die sich in eigene Programme einbauen lassen.

Nützliche Adressen

Routine	Adresse	Anmerkung
TXT_WIN_ENABLE	BB66	H = links, D = rechts, L = oben, E = unten
TXT_GET_WINDOW	BB69	Ergebnis in den Registern wie oben
TXT_CLEAR_WINDOW	BB6C	Löscht Textfenster des aktuellen Stroms
TXT_SWAP_STREAMS	BBB7	In B und C stehen beide Stromnummern
TXT_STR_SELECT	BBB4	Angewählte Ströme in Register A
TXT_OUTPUT	BB5A	Zeichen steht in Register A
TXT_WR_CHAR	BB5D	Zeichen steht in Register A
TXT_RD_CHAR	BB60	Liest beim Cursor Zeichen und speichert es in Reg. A
TXT_GET_CURSOR	BB78	H = Spalte, L = Zeile
GRA_SET_ORIGIN	BBC9	Aufruf mit DE = X, HL = Y (Koordinaten)
GRA_SET_WIDTH	BBCF	Aufruf mit DE = links, HL = rechts (Koordinaten)
GRA_SET_HEIGHT	BBD2	Aufruf mit DE = oben, HL = unten (Koordinaten)
SCR_SET_BASE	BC08	In A steht Hi-Byte der Basisadresse
SCR_SET_MODE	BC0E	In A steht Modusnummer (0, 1 und 2)

Wichtig: Für den Einsatz dieser Routinen sollten Sie sich unbedingt die „Amstrad Firmware Specification" besorgen.

Die Textanzeige steuert unter anderem das Lesen und Anzeigen von Zeichen, den Cursor und Fenster. Die Firmware von Schneider kann bis zu acht Fenster auf den Schirm bringen, jedes mit eigenen PEN- und PAPER-Farben, eigenem Cursor, transparentem oder undurchsichtigem Hintergrund und der Möglichkeit, Grafikzeichen darzustellen. Die Größe des Fensters wird mit TXT_WIN_ENABLE gesetzt, mit TXT_GET_WINDOW abgefragt und über TXT_CLEAR_WINDOW gelöscht.

Sie können die Parameter zweier „Ströme" mit TXT_SWAP_STREAMS auch gegeneinander austauschen. Die meisten Firmwareroutinen sprechen den jeweils aktiven Strom an, wobei TXT_STR_SELECT festlegt, welcher Strom aktiv ist.

Jedes Zeichen wird in einem Pixelquadrat von acht mal acht Pixeln ausgegeben. Wir haben die wichtigsten Anzeigeroutinen in der nebenstehenden Tabelle aufgeführt.

Die Bildschirmzeichen grenzen normalerweise aneinander und erscheinen an der aktuellen Cursorposition. Über einen Spezialmodus (Anwahl mit TXT_SET_GRAPHIC) können sie aber auch an der Position des Grafikcursors geplottet werden. Damit lassen sich Texte überlagern und an beliebigen Bildschirmpositionen darstellen.

Die Hauptroutine, TXT_OUTPUT, sichert zunächst alle Register und ruft dann TXT_OUT_ACTION auf. Die Routine, auf die dieses Modul zeigt, filtert die Steuerzeichen aus, bevor die darstellbaren Zeichen an TXT_WR_CHAR (bei grafischer Textausgabe GRA_WR_CHAR) übergeben werden. Da TXT_OUT_ACTION gepatcht werden kann, lassen sich auch andere Interpretationen der Steuercodes oder der Zeichendarstellung einsetzen.

Aktuelle Fenster

Beim Textbildschirm zeigt der Cursor auf die Position, an der das nächste Zeichen erscheinen soll. Einige Routinen können den Textcursor abschalten und andere Cursor an beliebige Bildschirmpositionen stellen oder löschen. Die wichtigsten dieser Routinen stehen ebenfalls in der Tabelle.

Mit dem Tabulatorprogramm können Sie Tabulatoren in gleichmäßigen Abständen setzen. Dabei wird zunächst mit TXT_GET_WINDOW die Größe des aktuellen Fensters festgestellt. Die Nummer der rechts außen liegenden Spalte befindet sich danach im Register D. Das Register wird umgewandelt und zeigt nun auf eine Position rechts dieser Spalte. Als nächstes wird mit TXT_GET_CURSOR die aktuelle Cursorposition geholt. Register H enthält nun die

Cursorspalte, die dann mit jeder möglichen Tabulatorposition verglichen wird, bis die nächste Tabulatoreinstellung gefunden ist. Nachdem diese Position bekannt ist, stellt ein Test sicher, daß das abzubildende Zeichenfeld auch über genug Platz verfügt. Ist dies der Fall, wird die Cursorspalte auf die nächste Tabulatoreinstellung gesetzt, wenn nicht, wird durch die Angabe einer Spalte außerhalb des aktuellen Fensters veranlaßt, daß der Cursor sich auf die darunterliegende Zeile bewegt.

Pixelweise

Jedem ASCII-Code von 0 bis 255 ist eine Matrix mit dem Bitmuster des jeweiligen Zeichens zugeordnet. Die Matrix besteht aus acht Bytes, wobei jedes Byte eine Zeile des Zeichens darstellt. Das Bild zeigt, wie die Zeichenmatrix für CHR$224 gespeichert ist.

Die Matrizen können über vier Firmwareroutinen festgelegt oder umdefiniert werden. Da alle Matrizen für die ASCII-Zeichen zwischen 0 und 247 standardmäßig im unteren ROM-Bereich untergebracht sind, lassen sie sich nicht beliebig umdefinieren. Die Zeichen 248 bis 255 werden automatisch ins RAM kopiert, um dort umgestellt zu werden. Mit der Routine TXT_SET_M_TABLE können Sie jedoch eine beliebige Anzahl Zeichen zwischen dem angegebenen Code und 255 umdefinieren und so einen völlig neuen Zeichensatz entwerfen.

Die Matrizen der umdefinierbaren Zeichen können Sie entweder mit einem Anwenderprogramm verändern oder über die Routine TXT_SET_MATRIX, die die acht Bytes, auf die das Registerpaar HL zeigt, in eine dem jeweiligen Zeichen entsprechende Matrix kopiert.

Mit TXT_GET_MATRIX können Sie die Adresse einer Zeichenmatrix feststellen. TXT_GET_M_TABLE holt das erste Zeichen und die Adresse einer Zeichentabelle.

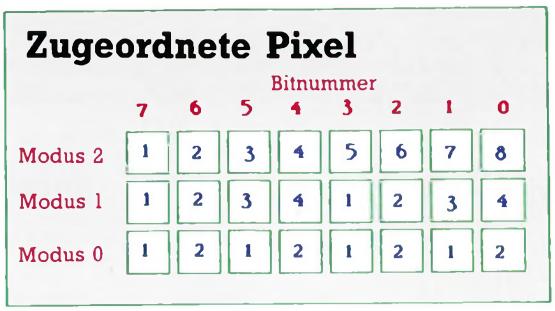
Unser „Spin Print Programm" arbeitet mit TXT_GET_MATRIX und TXT_SET_MATRIX, um Zeichen in eine von vier Richtungen zu rotieren. Dabei wird Zeichen 248 auf die Matrix des darzustellenden Zeichens umdefiniert, dann die Matrix von Zeichen 248 rotiert und schließlich mit TXT_OUTPUT dargestellt.

Der Grafikbildschirm läßt sich pixelweise steuern. Er arbeitet mit einem Grafikfenster, in dem die Vorder- und Hintergrundfarben unabhängig von denen des Textschirms gesetzt werden können.

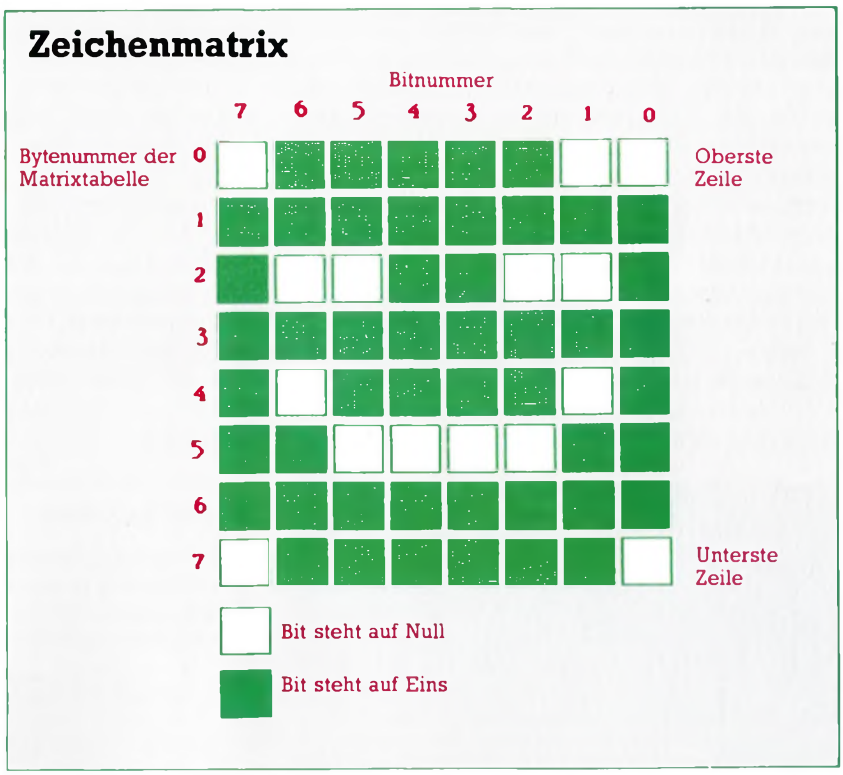
Es gibt drei Koordinatensysteme, mit denen sich Pixelpositionen festlegen lassen. In Modus 2 beträgt die Auflösung 640 mal 200 Punkte, in Modus 1 320 mal 200 und in Modus 0 160 mal 200. Diese Daten werden „Basiskoordinaten" genannt. Jeder Bildschirmpunkt hat eine eindeutige Adresse, über die er unabhängig vom aktuellen Modus angesprochen werden kann. Die Modi bestimmen dabei nur, wieviel Platz ein bestimmtes Pixel beim Setzen eines Punktes belegt. Modus 2 setzt ein Pixel, Modus 1 zwei und Modus 0 vier.

Obwohl der Bildschirm vertikal nur eine Auflösung von 200 Punkten besitzt, arbeitet die Firmware mit 400 Punkten, um das Verhältnis 2 : 1 beizubehalten. Jeder vertikale Punkt bezieht sich daher nur auf ein halbes Pixel, wobei die zwei Pixelhälften gleichzeitig gesetzt werden. So beziehen sich beispielsweise die beiden Pixelzeilen 10 und 11 auf die gleiche Bildschirmzeile. Durch dieses Verfahren hat jeder Bildschirmmodus eine Auflösung von 640 mal 400 Pixeln.

Die Fenstergröße wird mit GRA_SET_ORIGIN, GRA_SET_WIDTH und GRA_SET_HEIGHT gesetzt und mit GRA_GET_ORIGIN, GRA_GET_W_WIDTH und GRA_GET_W_HEIGHT abgefragt. Nach der Initialisierung eines Fensters beziehen sich die sogenannten „Anwender-Koordinaten" auf die neuen Daten. Der Grafikcursor läßt sich mit zwei Methoden positionieren: Beim „absoluten" Verfahren wird der Cursor auf die angegebene Anwenderkoordinate gesetzt, während die „relative" Methode den Cursor relativ zu seiner aktuellen



Die Bytes des Bildschirmspeichers enthalten je nach Anzeigemodus die Daten für bis zu acht Pixel. Das Bild zeigt, welche Bits sich in den unterschiedlichen Modi auf welche Pixel beziehen.



Zeichen werden in Acht-Bit-Matrizen gespeichert, in denen die einzelnen Bytes je eine Pixelzeile des Zeichens darstellen. Gesetzte Bits zeigen an, daß ein Pixel in der aktuellen PEN-Farbe dargestellt werden soll. Auf Null stehende Bits geben die PAPER-(Hintergrund-) Farbe an. Unser Bild zeigt die Matrix für das ASCII-Zeichen 244.

Position verschiebt. Die Tabelle zeigt die Positioniermethode.

Auf dem Grafikschirm lassen sich Pixel auf drei Arten plotten; einzeln, als Zeichen oder als Linien. Alle Plotroutinen können absolute Punkte oder Punkte relativ zur aktuellen Cursorposition anzeigen. Der Grafikcursor steht immer links von dem zuletzt geplotteten Punkt – außer wenn beim Plotten eines Zeichens die horizontale Position auf den Anfang des nächsten Zeichens verschoben und die vertikale Position beibehalten wird.

Das „Bildschirmpaket" ist ein Firmwarebereich, der maschinennahen Zugang zum Bildschirm bietet. Er steuert die Bildschirmadressierung, das Setzen von Farben und Modi, die Codierung und Decodierung, das Scrollen und vieles andere.

Der Bildschirmspeicher ist ein RAM-Block von 16 KByte, der auf jeder der vier 16K-Grenzen des Speichers anfangen kann. Sie sollten aber nur die Blöcke mit den Anfangsadressen $4000 und $C000 nehmen, da die anderen beiden Blöcke Firmwarebereiche enthalten und überschrieben werden könnten. Der Bildschirm kann mit SCR_SET_BASE zwischen zwei Speichern umgeschaltet werden. Mit dieser Technik läßt sich ein Bildschirminhalt anzeigen, während der andere vorbereitet wird. Durch Umschalten wechselt die Anzeige.

Jedes für den Bildschirm eingesetzte Speicherbyte enthält die Informationen für acht Anzeigepositionen. Dies mag je nach dem (mit SCR_SET_MODE gesetzten) Modus zwei, vier oder acht Pixeln entsprechen.

Das Bild mit den Pixelzuordnungen zeigt, wie die Bits eines Bildschirmbytes in verschiedenen Modi die Pixel festlegen. Zwar sieht das System auf den ersten Blick kompliziert aus, doch erscheinen durch die Rotation der Bytes automatisch die Bits für das nächste Pixel. Für das Setzen eines Pixels in einem Byte brauchen Sie nur über eine Maske das links außen liegende Bit zu setzen und das Byte so lange zu rotieren, bis das Bit an der gewünschten Position steht.

Maskenhaft

Den schnellsten Bildschirmzugriff haben Sie, wenn Sie den Bildschirmspeicher direkt ansprechen und nicht mit dem Text- oder Grafikschirm arbeiten. Hier bietet das Bildschirmpaket etliche Routinen, die diese Aufgabe erleichtern.

Unser „Schnellplot-Programm" zeigt, wie mit diesen Routinen Punkte auf den Schirm gebracht werden. Das Programm darf nicht mit einem Grafikschirm eingesetzt werden, der mit Fenstern arbeitet.

Die Routine fragt zunächst nach der Adresse des Bytes mit den Daten für die angegebene Bildschirmposition und dann nach der Maske, mit der alle Pixel eines Bytes mit der angegebenen Farbe definiert werden. Die Bearbeitung der Maske setzt die gewünschte Position zunächst auf INK Null und dann auf die neue INK-Farbe.

Plot mit 128K

Der PLOT-Befehl des Schneider 6128 wurde gegenüber dem CPC 464 etwas verfeinert:

Modus	Aufgabe
0 Normal	– setzt die Pixel auf die neue INK-Farbe
1 XOR	– setzt die Pixel auf die neue oder (XOR) aktuelle Farbe
2 AND	– setzt die Pixel auf die neue und (AND) die aktuelle Farbe
3 OR	– setzt die Pixel auf die neue oder (OR) die aktuelle Farbe

Besonders der XOR-Modus (exklusives OR) ist interessant, da Sie damit Linien plotten und löschen können, ohne die Hintergrundfarbe zu beeinflussen. Bei einer roten Hintergrundfarbe (PAPER 3) erzeugt das Plotten einer Linie (PEN 4) eine Linie in Magenta (PEN 7). Zum Löschen brauchen Sie die Linie nur ein zweitesmal mit PEN 4 zu plotten.

Diese Methode eignet sich besonders für die Sprite-Steuerung. Auf dem 6128 werden die Plot-Modi durch einen zusätzlichen Parameter des PLOT-Befehls angegeben. Diese Möglichkeit gibt es im BASIC 1.0 des CPC 464 noch nicht. Sie brauchen dort nur die Firmwareroutine bei &BC59 mit der Modusnummer im Register A aufzurufen.

Tabulatoren

Das Programm bewegt den Textcursor des aktuellen Fensters in die nächste Tabulatorenspalte. Es gibt keine Einsprungsbedingungen. Die Routine sichert alle Register.

```
tab:            equ     15      ; tab stop column
get.window:     equ     #bb69   ; TXT_GET_WINDOW
get.cursor:     equ     #bb78   ; TXT_GET_CUSOR
set.column:     equ     #bb6f   ; TXT_SET_COLUMN
;
        push    af
        push    hl
        push    de
        call    get.window      ; get window size
        inc     d               ; logical rt col
        inc     d               ; right col limit
        call    get.cursor      ; get cursor posn
        ld      a, 1            ; start of line
;
loop:   add     tab             ; get next tab col
        cp      h
        jp      p, loop
;
        ld      h, a            ; save it
        add     tab             ; room for new col?
        cp      d               ; test for limit
        jr      nc, setcol      ; no,force newline
        ld      a, h            ; get back new col
setcol:
        call    set.column      ; set new column
        pop     de
        pop     hl
        pop     af
        ret
```

Spin Print

Das folgende Programm gibt ein Zeichen aus, dessen Zeichenmatrix um einen von vier Winkeln rotiert wurde. Dafür wird zunächst die Adresse der Zeichenmatrix festgestellt und Zeichen 248 (normalerweise der erste Eintrag in der Tabelle der anwenderdefinierten Zeichen) umdefiniert.

```
;
; Entry:        IX+0 = Character to display
;               IX+2 = Desired angle 1..4
;                    1 = upwards
;                    2 = backwards
;                    3 = downwards
;                    4 = normal
; Exit:
;               Carry True  - Character printed
;               Carry False - Failed to print char
;               AF, BC, DE, HL corrupt
;
txt.get.matrix: equ     #bba5
txt.set.matrix: equ     #bba8
txt.output:     equ     #bb5a
;
        org     8000h
;
start:  ld      a, (ix+0)       ; read character
        call    txt.get.matrix  ; get matrix addr
;
; Copy matrix into user definable character 128
; then find its address
;
        ld      a, 248          ; char to redefine
        call    txt.set.matrix  ; redefine matrix
        ret     nc              ; abort if unable
;
        ld      a, 248          ; char to rotate
        call    txt.get.matrix  ; find matrix
;
; Rotate the character matrix anti-clockwise
; in 90 degrees steps
;
        ld      (toprow), hl    ; save address
        ld      b, (ix+2)       ; read angle
;
lop3:   ld      d, 8            ; loop for 8 rows
;
lop2:   ld      e, 8            ; loop for 8 bits
;
lop1:   rlc     (hl)            ; next bit in c
        rla                     ; save in new row
        inc     hl              ; go to next row
        dec     e
        jr      nz, lop1        ; go for next row
;
        push    af              ; stack new row
        dec     d
        ld      hl, (toprow)    ; go for next bit
        jr      nz, lop2
;
; loop to unstack the new matrix
;
        ld      e, 8
lop4:   pop     af              ; get next row
        ld      (hl), a         ; move to matrix
        inc     h               ; go to next row
        dec     e
        jr      nz, lop4
;
        ld      hl, (toprow)
        djnz    lop3            ; again if needed
;
; Finally display the character on screen
;
        ld      a, 248          ; char to print
        call    txt.output
        ret                     ; finished
;
toprow: defw    0               ; start of matrix
```

Text – einmal andersherum
Mit unserem Programm zur Zeichenrotation können Sie Text vertikal, horizontal oder auch auf dem Kopf stehend ausgeben.

Ein Schnell-Plot-System

Diese Routine funktioniert in allen Bildschirmmodi. Per Firmware steuert sie eine Speicherstelle mit Informationen über den Bildschirmpunkt an und stellt so das neue Pixel dar.

```
; Entry:  HL contains the physical Y co-ordinate
;         (0-199)
;         DE contains the physical X co-ordinate
;         (0-639)
;         A contains the ink to set the point to
;
; Exit:   All registers preserved
;
dotpos:   equ  #bc1d      ; SCR_DOT_POSITON
encode:   equ  #bc2c      ; SCR_INK_ENCODE
;
plotter:
          push de   ; save the registers
          push hl
          push bc
          push af
;
          call dotpos     ; find the memory byte
          pop  af         ; get the ink back in a
          push af
;
; HL now contains the address of the byte
; and C contains the mask required to set only the
; required pixel on
;
          call encode     ; find the ink mask
;
; The mask to set all pixels to new ink is in a
;
          and  c          ; only want this pixel
;
          ld   b, a       ; save the new ink number
          ld   d, (hl)    ; get the current setting
          ld   a, c       ; get the pixel mask
          cmp  a          ; invert it
          and  d          ; set the pixel to ink 0
;
          or   b          ; mask in the new colour
          ld   (hl), a    ; and store it out
;
          pop  af         ; restore the registers
          pop  bc         ;
          pop  hl
          pop  de
;
          ret
;
```

Befehlsempfänger

Wir behandelten die Datenübertragungsbefehle und einfache Arithmetikanweisungen des Motorola 68000. Bevor wir weitergehen, müssen wir herausfinden, wie die Vergleichsbefehle des 68000 arbeiten.

Der Befehl CMP (compare – vergleichen) ist der SUB-Anweisung sehr ähnlich. Sehen Sie sich folgende Programmstruktur im Pseudo-Code an:

```
if Eingabezeichen = 'N' then
   Array-löschen
end
```

Die IF-Entscheidung wird in der Implementierungsphase des Programms als

```
CMP.B #'N',D0     Vergleiche Eingabebyte
                  mit 'N'
BNE UNGLEICH      Auf UNGLEICH verzweigen,
                  wenn Eingabebyte
                  ungleich 'N'
```

Der CMP-Befehl subtrahiert zwar die Quelle vom Ziel, verändert aber nur die Bedingungscodes. Der Vorgang hat keinen Einfluß auf das Ziel (wie beispielsweise SUB). Wenn das Ergebnis der Subtraktion von D0 (das heißt 'N') nicht gleich Null ist, dann veranlaßt die bedingte Verzweigung – BNE – eine Verzweigung auf das Label UNGLEICH.

Hier weitere Beispiele mit anderen Datenattributen:

```
CMP.W SPEED,D3    Vergleiche den Wort-
                  inhalt der Speicher-
                  stelle SPEED mit D3
CMP.L D1,D2       Vergleiche die beiden
                  Langwörter in D1 und
                  D2
```

Die Ziele dieser beiden Beispiele sind Datenregister. Dabei kann zur Adressierung der Quelle jeder beliebige Modus eingesetzt werden. Wenn Sie eine andere Art Ziel adressieren wollen, müssen sie andere CMP-Befehle verwenden, zum Beispiel:

```
CMPA BETTY,A3     Vergleiche den Inhalt
                  der Speicherstelle
                  BETTY mit A3
```

Dabei lassen sich zwar alle Adressierungsarten für Quellen einsetzen, doch sind im unmittelbaren Format nur datenverändernde Modi möglich:

```
CMPI #3, (A4)     Vergleiche, ob A4 als
                  Pointer auf 3 zeigt
```

Schließlich verdient noch die Variante CMPM besondere Aufmerksamkeit. CMPM (A2)+, (A3)2 vergleicht beispielsweise den Inhalt der Speicherstellen, auf die A2 und A3 zeigen, und nach-inkrementiert dann die Pointer. Damit kann ein einziger Befehl die gespeicherten Schlüsselwörter mit den Zeichen eines Eingabebuffers vergleichen:

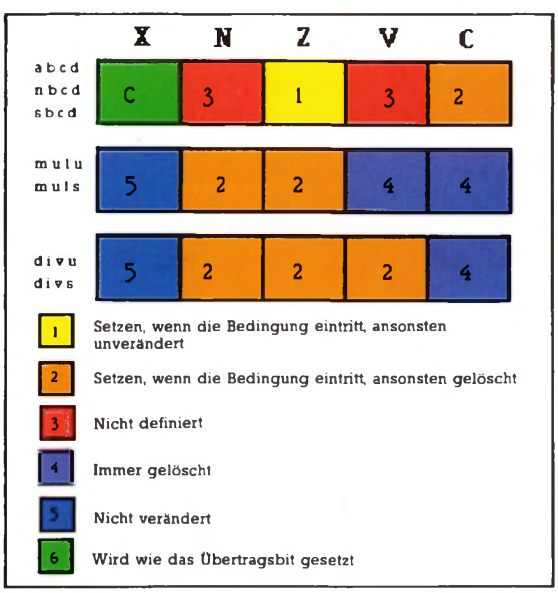
```
LEA KEYS,A2       Den ersten Pointer auf
                  die Schlüsselwörter
                  setzen
LEA BUFFERS,A3    Bufferpointer anlegen
CHECK CMPM.B      Die beiden Speicher-
                  stellen vergleichen
BEQ CHECK         Prüfen, bis ungleich
```

Das gesamte Programm belegt nur sieben Wörter.

Bei CMPM ist nur die Adressierung mit Nach-Inkrementierung möglich. Für andere Formate können Sie einen Operanden in ein Datenregister laden und dann CMP einsetzen.

NEG und EXT sind zwei einfache, aber sehr wichtige Arithmetikbefehle. NEG (Negation) zieht das Datenregister des Operanden von Null ab, das heißt, es bildet aus dem Registerinhalt den negativen Wert des Zweierkomplements (hier sind keine weiteren Adressierungsmethoden möglich). Wenn Sie mit D0 = 1111 1010 (dezimal –6) NEG.B D0 ausführen, dann enthält D0 am Ende 0000 0110 (dezimal 6). Mit diesem Befehl wird oft der absolute Wert von Datenoperanden gebildet: Zuerst wird festgestellt, ob ein negativer Wert vorliegt, und dann negiert. Eine erweiterte Form dieses Befehls (NEGX) schließt das X-Bit in den laufenden Vorgang mit ein.

Mit EXT (Vorzeichenerweiterung) wird das Vorzeichenbit des Datenoperanden auf die nächstgrößere Operandengröße erweitert.



Das nebenstehende Bild zeigt die Auswirkungen der DIV-, MUL- und BCD-Befehle auf das Statusregister. Fehlt ein Bedingungscode, so bedeutet das jedoch nicht, daß das entsprechende Flag auch gelöscht ist. Wenn Sie einen (von einem vorangehenden Ablauf gesetzten) Bedingungscode testen, erhalten Sie falsche Werte.

EXT.W D0 (mit D0 = 11110101) ergibt FFFA (hex). Dieser Befehl läßt sich gut für Zahlen mit doppelter Genauigkeit einsetzen und besonders für große Operanden bei Multiplikations- und Divisionsbefehlen.

Bevor wir uns den komplizierteren arithmetischen Anweisungen zuwenden, müssen wir untersuchen, was die binären Bitmuster der Datenbytes überhaupt bedeuten. So könnten wir die Zahl 1001 0110 (z. B. in D0 gespeichert) als Ganzzahl mit dem Werk −106 ansehen (der Wert entsteht, wenn Sie die Zahl umkehren und Eins addieren). Wenn die Zahl jedoch nicht vorzeichenbehaftet ist und der gesamte Binärbereich von positiven Ganzzahlen belegt wird, dann ergibt sich hexadezimal 96 (dezimal 150). Zahlen ohne Vorzeichen sind praktisch, wenn nur ein großer positiver Zahlenbereich gebraucht wird. So läßt sich beispielsweise der Adreßbereich eines Computers als ein Bereich positiver Zahlen ohne Vorzeichen ansehen. Wenn Sie diese Zahlen nicht mit Operatoren im Zweierkomplement bearbeiten, gibt es keine Probleme.

Illegale Bitfolgen

Und es gibt noch eine weitere Interpretation der Bitmuster: das BCD-System (binär codierte Dezimalzahlen). Bei diesem praktischen Datencode werden Dezimalzahlen in jeweils vier Bits codiert. Unser Beispiel (D0 = 1001 0110) würde daher den BCD-Wert von 96 ergeben (1001 = 9 und 0110 = 6).

Selbst große Zahlen lassen sich leicht in BCD-Zahlen umwandeln und umgekehrt. Die Dezimalzahl 9631 sieht im BCD-Code so aus: 1001 0110 0011 0001.

Ebenso einfach wird die Rechengenauigkeit definiert. Dabei sind beim BCD-Code bestimmte Bitfolgen illegal. Da BCD nur die Zahlen von 0 bis 9 codiert, gibt es keine Verwendung für die Codes zwischen 1010 (dezimal 10) und 1111 (dezimal 15).

Bei der Behandlung der Arithmetikbefehle des 68000 werden Sie sehen, wie wichtig dabei die Interpretation der Bitmuster ist. Untersuchen wir zunächst den Ablauf einer binären Multiplikation. Wenn zwei 16-Bit-Operanden multipliziert werden, kann das Bitmuster des Ergebnisses 32 Bits lang sein. Für eine Wortlänge von n ist das Produkt der größten Zahl, $2^{(n-1)}$, die doppelte Länge des ursprünglichen Wortes oder $2^{2(n-1)}$.

Binäre Multiplikationsbefehle haben daher zwei 16-Bit-Operanden und ein 32-Bit-Ergebnis. Die Verarbeitung von Zahlen mit und ohne Vorzeichen geschieht über zwei unterschiedliche Befehle – MULU (multiplizieren ohne Vorzeichen) und MULS (multiplizieren mit Vorzeichen). Beide Befehle multiplizieren die Operanden und erzeugen in dem Zielregister ein Ergebnis im 32-Bit-Format, wobei der Operand nur mit Datenadressierungsarten angesprochen werden kann, damit der Computer den Befehl ausführt.

MULU #20,D0 für D0 = XXXX 0003 (X kann 0 oder 1 sein) ergibt D0 = 0000 003C. Dabei wird das gesamte 32-Bit-Datenregister eingesetzt, obwohl es in diesem Fall nicht nötig ist.

Auf ähnliche Weise erhalten Sie bei MULU #$10,D0 für D0 = XXXX FFFF das Ergebnis D0 = 000F FFF0. Der Befehl MULS ergibt D0 = FFFF FFF0, da sich das vorzeichenbehaftete Ergebnis auf 32 Bits erstreckt. Die Resultate lassen sich leicht prüfen, da der Multiplikator 10 (Hex) einer vierfachen Linksverschiebung oder einer Multiplikation mit 16 entspricht.

Auch die Bedingungscodes sind bei der Multiplikation wichtig. So werden die Flags N und Z je nach Ergebnis gesetzt und die Bits V und C auf Null gestellt. Obwohl Sie bei Operanden mit Wortlänge und einem Langwortergebnis nie einen Überlauf erhalten können, lohnt sich die Feststellung, ob das Ergebnis die Wortlänge überschreitet (damit es, falls nötig, auf die von Ihnen gewünschte Wortlänge gekürzt werden kann).

Das folgende Programmbeispiel zeigt, wie der Befehl MULS für die Umwandlung von dezimalen ASCII-Codes in Binärwörter eingesetzt wird. Als erstes muß das Eingabezeichen in Binärformat umgewandelt und dieses Bitmuster dann in einen binären Akkumulator addiert werden. Vor der Addition muß jedoch sichergestellt sein, daß die vorige Eingabe mit 10 multipliziert wurde (da sie im Dezimalformat erfolgt). Hier der Code:

```
SUB.B #'0',D0   Dezimalen Zeichencode in
                Binärformat wandeln
MULS #10,D5     Die alte Summe mit dezimal
                10 multiplizieren
ADD.W D0,D5     Neuen Wert addieren
```

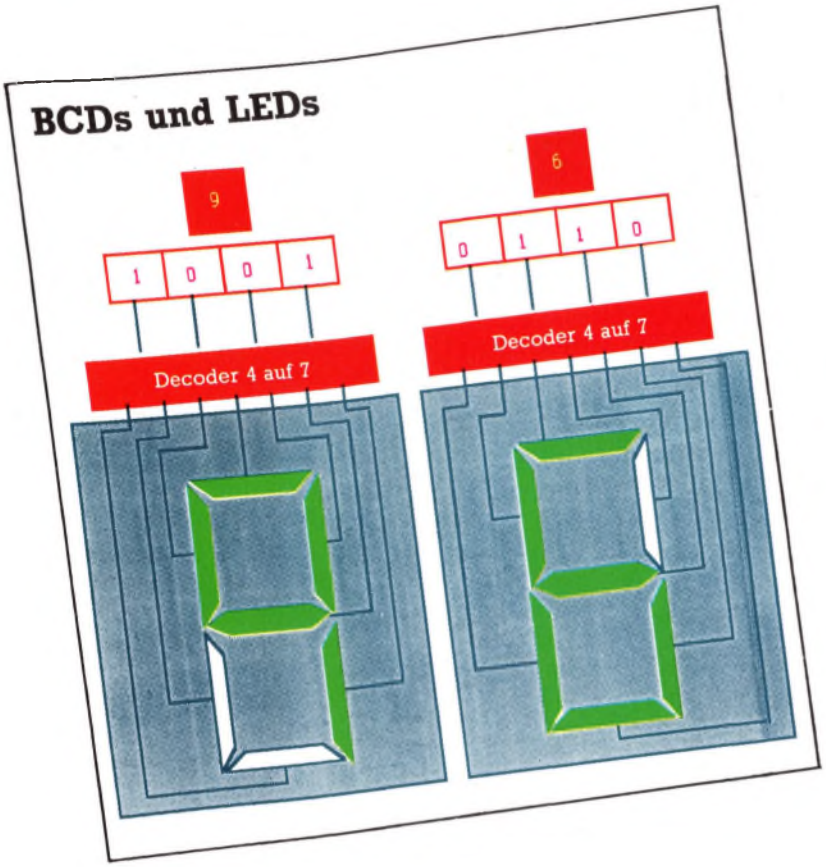
In diesem Beispiel befindet sich das Eingabezeichen in D0 und das neue Binärwort mit dem binären Zeichencode in D5.

Gute Vorzeichen

Auch für die Subtraktion gibt es zwei Befehle – DIVU für Zahlen ohne Vorzeichen und DIVS für Zahlen mit Vorzeichen. Beide Befehle nehmen die Ganzzahl mit 32-Bit-Format im Zieldatenregister und teilen sie durch den Quelloperanden im 16-Bit-Format. Das Ergebnis wird in den niederwertigen 16 Bits des Zieldatenregisters gespeichert, während der Rest in den höherwertigen 16 Bits erscheint. Bei D0 = 0000 0005 (Hex) ergibt

```
DIVU #3,D0
```

D0 = 0002 0001 (das heißt 1 Rest 2). Da der Zieloperand 32 Bits lang ist, sollte, falls nötig, zuvor das entsprechende Langwort gelöscht oder mit EXT.L eine Langworterweiterung angegeben werden. Da Überläufe auftreten können (das Ergebnis der Teilung eines 32-Bit-Wortes durch Eins kann durchaus das 16-Bit-Format überschreiten), muß in bestimmten Fäl-



Der Befehlssatz des 68000 enthält nicht nur Codes für Zahlen mit und ohne Vorzeichen, sondern auch für die Verarbeitung von BCD-Formaten (binär codierte Dezimalzahlen). Diese Zahlen sind besonders praktisch, wenn die größtmögliche mathematische Genauigkeit gefordert ist. Auch in anderen Anwendungsbereichen, wie beispielsweise in der Steuerung von siebenteiligen LED-Anzeigen, hat das BCD-Format Vorteile. So ist die (im Bild gezeigte) Decodierung von vier auf sieben Bits einfacher als die Umwandlung von Binärzahlen.

len von Ihnen auch das Überlaufbit berücksichtigt werden.

Hier ein einfacher Algorithmus, der den Durchschnitt aller Zahlen einer Wortliste bildet. Die Liste endet mit einer Null:

```
                ORG $1000
        LEA     LIST1,A0        Adresse
POINTER                         anlegen
                                Ergebnisregister
        CLR.L   D0              löschen
        CLR.L   D1              Schleifenzähler
                                löschen
LOOP    ADDQ    #1,D1           Wortzahl zählen
        ADD.W   (A0)+,D0        Summe bilden
        TST     (A0)            Auf Ende der
                                Liste prüfen
        BNE     LOOP
        DIVU    D1,D0           Summe durch
                                Zähler teilen
        TRAP    0
LIST1   DC.W    1,2,3,4,0
```

Im Initialisierungsteil des Programms werden Langwörter gelöscht, da die Summe das 32-Bit-Format hat. D1 ist der Zähler, und A0 zeigt auf das Wort, das mit nach-inkrementierter Adressierung auf D0 addiert wird. Danach wird das nächste Wort von LIST1 mit TST getestet. Dieser Befehl setzt nur die Bedingungscodes für BNE, ändert den Operand aber nicht.

Der Befehl TST ist nötig, da das vorangegangene ADD die Bedingungscodes nach dem Ergebnis der Addition des Datenoperanden auf D0 setzt, nicht aber nach dem Wert der Daten, die über den Pointer A0 geladen werden. Der Befehl BNE (verzweige bei ungleich Null) veranlaßt einen Rücksprung auf LOOP, wenn das nächste Listenelement nicht Null ist. Nach dem Auftreten einer Null enthält D0 die Summe und D1 die Zahl der Elemente, die nicht Null sind.

Nach vier Schleifendurchgängen stehen in den Datenregistern folgende Werte:

D0 = 0000 000A (oder dezimal 10)
D1 = 0000 0004

Nach Ausführung von DIVU enthält D0 den Wert 0002 0002 (10 / 4 = 2 Rest 2).

Ein letzter Hinweis: Eine Teilung durch Null löst ein „Trap" (ein Softwareinterrupt im Systemmonitor) aus, da Unendlichkeit sich in 16 Bits nicht darstellen läßt. Sie können statt des Trap auch prüfen, ob der Divisor Null ist. Hier ein Beispiel:

```
TST   D1
BEQ   ERROR
DIVU  D1,D0
```

Wie praktisch das BCD-Format für die Codierung von Dezimalzahlen sein kann, hatten wir schon erwähnt. Hierbei entspricht eine legale BCD-Stelle einer Hexadezimalstelle (4 = binär 0100 = 4 hex = 4 BCD). BCD-Konstanten werden daher auf die gleiche Weise gesetzt wie Hex-Konstanten:

```
MOVE.B #$54,D0 speichert BCD 54 in D0
```

Doch hier endet die Ähnlichkeit auch schon. Wenn Sie zwei BCD-Stellen mit binärer Arithmetik addieren, erhalten Sie ein falsches Ergebnis

```
0100    1001    (BCD 49) binär addiert
0000    0001    (BCD 1)
---------------
0100    1010
```

Dabei entsteht für die niederwertige BCD-Stelle ein illegaler BCD-Code. Der 68000 bietet für diesen Zweck einen Satz von BCD-Befehlen: ABCD (Addition), SBCD (Subtraktion) und NBCD (Negation).

Der BCD-Befehl ABCD addiert das Quellenbyte (je zwei BCD-Zahlen) mit dem Bit X auf das Zielbyte und speichert dort auch die Summe. Hier sind als Adressierungsarten jedoch nur Datenregisterpaare und vor-dekrementierte Datenregisterpaare erlaubt. Bei D0 = 44 BCD und D1 = 01 BCD und nach Ausführung des Befehls

```
ABCD D0,D1
```

enthält D1 45 BCD. Wenn das Bit X des SR gesetzt ist (entspricht dem Übertrag von BCD) würde das Ergebnis 46 lauten. In ähnlicher Weise ergibt die Addition von 1 auf 99 in D0 den Wert 00 mit dem gesetzten Bit im Statusregister. Zwar lassen sich nur Byteoperanden angeben, doch ermöglicht Bit X eine hohe Rechengenauigkeit, da Überläufe in höherwertige Bytekomponenten übernommen werden.

Die Bedingungscodes X, C und Z werden bei allen BCD-Befehlen gesetzt. Beachten Sie jedoch, daß der Befehl NBCD (negiere BCD) beim Zieloperanden datenverändernde Adressierungsarten zuläßt.

Harte Bedingungen

In der letzten Folge hatten wir die Multiplikation und Division des 68000 beschrieben und dabei auch die BCD-Befehle für Addition und Subtraktion behandelt. Wir wenden uns nun den Logikbefehlen zu und untersuchen danach die Anweisungen für Verschiebung, Rotation und Programmverzweigung.

Sehen wir uns zunächst die Logikbefehle des 68000 an. Die Anweisung AND verknüpft den Quelloperanden durch ein logisches AND mit dem Zieloperanden, legt das Ergebnis im Ziel ab und setzt – falls nötig – die Bits N und Z. Es sind viele Adressierungsarten möglich, doch die Operanden müssen Datenregister sein. Weiterhin sind Datenattribute zulässig. Wenn D0 = 1010 1010 und D1 = 1111 0000, dann ist nach dem Befehl

AND D1,D0

das Datenattribut D0 = 1010 0000.

Unser Beispiel maskiert die niederwertigen vier Bits von D0 aus, da die entsprechenden Bits von D1 auf 0 stehen. Die höherwertigen Bits erscheinen im Ziel, weil die Maskenbits dieser Positionen gesetzt sind. Der Befehl „ANDI" kann den Quelloperand unmittelbar adressieren, dabei sind im Zieloperand datenverändernde Modi möglich. Unser voriges Beispiel sieht mit ANDI so aus:

ANDI #$F0,D0

Neben OR steht ORI zur Verfügung, für das exklusive OR lassen sich EOR und EORI einsetzen, und auch das logische NOT gibt es. Diese Befehle funktionieren wie AND und sprechen auch die gleichen Bedingungscodes an. Die Adressierung einzelner Bits wird oft für die Steuerung von „Flag"-Bits verwandt.

Für Datenoperanden, die aus einem einzelnen Bit bestehen, bietet der 68000 vier Bitanweisungen, die sich statt der Logikbefehle einsetzen lassen. Die Bitanweisungen testen den Status der angegebenen Bits, die entweder in einem Langwort (Datenregister) von 0 bis 31 durchnumeriert sind oder sich in einem Speicherbyte befinden. Die Anweisungen setzen je nach dem Status des Bits das Z-Bit des SR. Z ist damit ein umgekehrter Ein-Bit-Speicher des angegebenen Bits. Bei D0 = XXXX XXXX XXX1 0000 (Binärformat – X ist 0 oder 1) testet

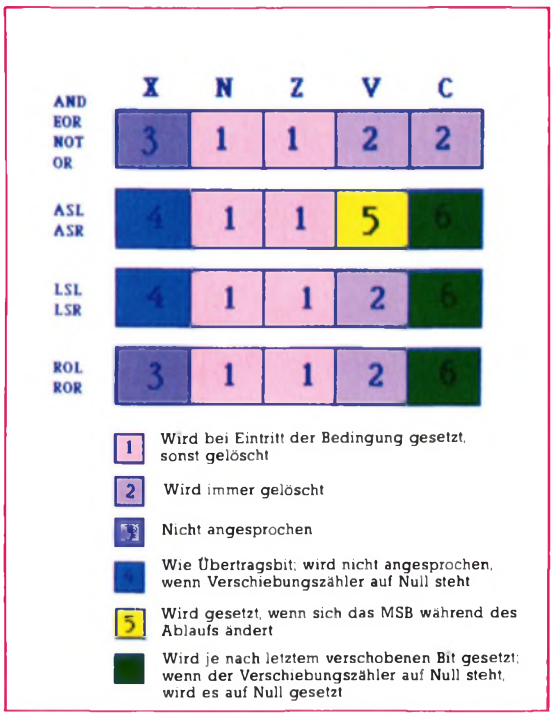
BTST #4,D0

das Bit vier und setzt 2 auf Null, wenn das entsprechende Bit nicht auf Null steht:

BTST #3,DO

setzt Z in diesem Fall auf Eins. Die anderen Bedingungscodes werden nicht verändert.

Folgende Befehle ändern das getestete Bit:
BSET setzt das Bit nach dem Test
BCLR löscht das Bit nach dem Test



Die logische Verschiebung stellt je nach Verschiebungsrichtung Nullen in Bit 0 oder 15. Während LSL wird Bit V auf Null gesetzt, während nach LSR Bit N auf Null steht. Beide Abläufe stellen das letzte verschobene Bit in C und X.

BCHG ändert das Bit nach dem Test
Wenn Sie mit den obenstehenden zahlreichen Beispieldaten

BCHG #4,D0

ausführen und D0 auf 1 gesetzt ist, erhalten Sie D0 = XXXX XXXX XXX0 0000, wobei Z auf Null gesetzt wird und damit den Zustand vor der Änderung anzeigt.

Beachten Sie, daß alle Befehle für das bitweise Testen und Setzen nur aus einer Anweisung bestehen. Dies kann besonders bei Mehrplatzanwendungen wichtig sein, in denen ein Interrupt zwischen Testen und Setzen zu unvorhersehbaren Ergebnissen führen kann. Logikbefehle haben den Vorteil, daß Tests keine Abläufe auslösen und lassen sich daher problemlos für die Bitverarbeitung einsetzen.

Sehen wir uns zunächst einmal an, was passiert, wenn ein Bitmuster nach links oder rechts verschoben wird. Wenn D0 = 0000 0000 0000 1000, dann erhalten wir bei einer dreifachen Rechtsverschiebung 0000 0000 0000 0001. Da der Inhalt von D0 dabei durch acht (2^3) geteilt wurde, entspricht die Rechtsverschiebung um eine Position einer Teilung durch Zwei. Umgekehrt ist die Linksverschiebung dementsprechend eine Multiplikation.

Bei der arithmetischen Rechtsverschiebung (ASR) wird Bit 15 in Bit 14 kopiert, der Inhalt von Bit 15 aber nicht verändert. Auf diese Weise bleibt das Vorzeichenbit erhalten. Bit 0 wird in Bit C und X des Statusregisters kopiert. Bei der Linksverschiebung werden Nullen in Bit 0 gestellt, und Bit V wird gesetzt, wenn sich der Inhalt von Bit 15 ändert.

Bei der Rechtsverschiebung haben wir vorausgesetzt, daß auf der linken Seite (bei der Linksverschiebung auf der rechten Seite) automatisch Nullen eingesetzt werden. Wenn das Vorzeichen der Zahl erhalten bleiben soll, muß dies bei einer Rechtsverschiebung berücksichtigt werden. Bei D0 = 1111 1111 1111 0000 (dezimal −16) ergibt eine dreifache Rechtsverschiebung D0 = 1111 1111 1111 1110 (dezimal −2). In diesem Fall wurden links Einsen eingesetzt, um die negative Zahl zu erhalten. Im allgemeinen setzt die Rechtsverschiebung Bits ein, die dem Vorzeichenbit (dem höchstwertigen Bit des Datenoperanden) entsprechen, während die Linksverschiebung Nullen in das niederwertige Bit des Wortes stellt. Verschiebungen, die das Vorzeichen des Operanden beibehalten, heißen „arithmetische Verschiebungen".

Der 68000 bietet die Befehle ASL (arithmetische Linksverschiebung) und ASR (arithmetische Rechtsverschiebung). Wenn das Ziel ein Datenregister ist, können Sie bis zu acht Bits unmittelbar verschieben, bei einem Langwort wird ein anderes Datenregister als Zähler eingesetzt und so bis zu 31 Bits verschoben. So ist

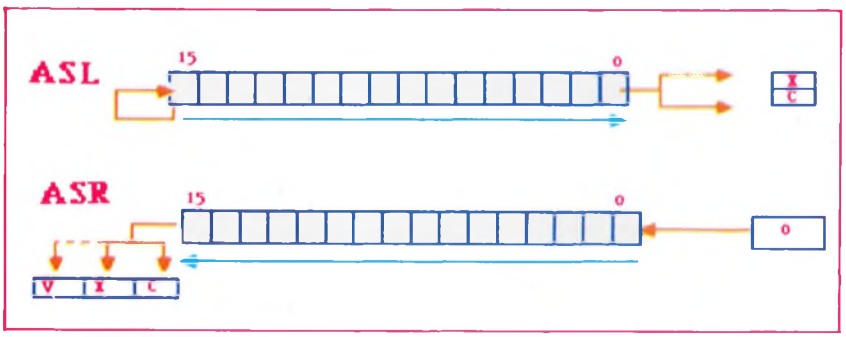
```
ASR #8,D0
```

die größte Verschiebung, während

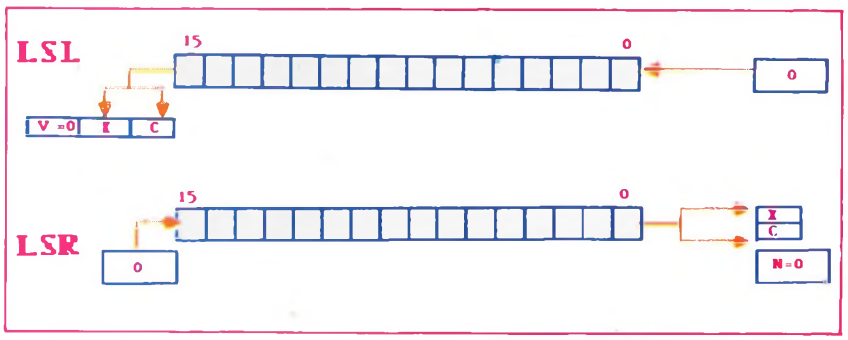
```
ASL D1,D0
```

D0 um soviel Positionen nach links verschieben kann, wie D1 Stellen hat (d. h. bis zu 31). Speicherstellen lassen sich jedoch nur um eine Position verschieben.

Das Bild zeigt, daß alle Bedingungscodes des SR angesprochen sind: C und X werden auf das letzte aus dem Operanden geschobene Bit gesetzt, V zeigt bei Linksverschiebungen die Veränderung des Vorzeichenbits an, während N und Z je nach Ergebnis gesetzt werden.



Die Befehle für Logik, Verschiebung und Rotation beeinflussen den Inhalt des Statusregisters. Die arithmetische Verschiebung unterscheidet sich von der logischen nur durch den Gebrauch des V-Flags, das mögliche Veränderungen im Vorzeichenbit des Operanden anzeigt.

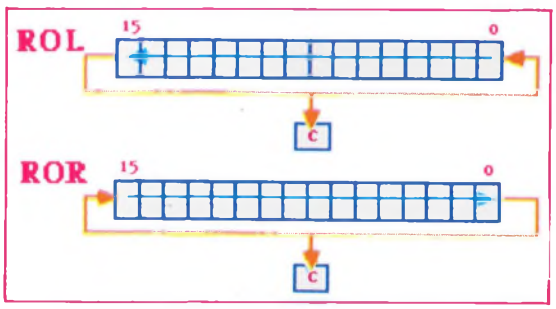


MULT-Vorgänge

Sie werden sich fragen, warum arithmetische Verschiebungen so wichtig sind, da der 68000 doch ausreichend mit Multiplikation- und Divisionsbefehlen ausgerüstet ist. Der Grund ist die Ausführzeit. Ein MULT-Vorgang braucht 70 Taktzyklen, während ASL oder ASR nur 6+2n Zyklen dauert (n ist die Zahl der Verschiebungen). Die Dauer einer arithmetischen Verschiebung reicht von acht Zyklen für die Verschiebung einer Position bis zu 68 Zyklen für alle 31 Positionen: Eine geringe Zahl von Verschiebungen ist daher weit schneller als Multiplikations- oder Divisionsbefehle – im Extremfall ist eine einfache Rechtsverschiebung um das Neunzehnfache schneller als die Teilung durch 2.

Es gibt aber auch die „logischen Verschiebungen" LSR und LSL, die das Vorzeichen des Datenoperanden nicht berücksichtigen und in den freigewordenen Stellen Nullen einsetzen. Auch hier gelten die Adreßbeschränkungen für arithmetische Verschiebungen. Das Bit V wird immer auf Null gesetzt. Logische Verschiebungen werden oft für das Setzen oder Testen von Datenoperanden eingesetzt, die kleinere Untergruppen oder Felder enthalten.

Die ROTATE-Befehle lassen den Operanden kreisen, indem sie bei ROR Bit 0 in Bit 15 stellen und umgekehrt (bei ROL). Das „übergestellte" Bit wird dabei in das C-Bit des Statusregisters kopiert.



Schließlich bietet der 68000 noch eine Gruppe von Rotationsbefehlen, die eigentlich Relikte aus einer Vergangenheit sind, in der die einzelnen Bits des Datenoperanden über die Rotation durch das Bedingungsflag getestet wurden. Im wesentlichen hat das breite Spektrum der Bitbefehle diese Aufgabe übernommen, wobei die neuen Befehle die gleiche Zeit brauchen wie die alten. Wir wollen daher nur ein Beispiel für die Rotation anführen. ROR

#3,D0 rotiert den Inhalt von D0 um drei Positionen nach rechts, setzt das Bit C auf das letzte verschobene Bit und bringt das niederwertige Bit an die höchstwertige Stelle.

Die Befehle zur Programmsteuerung legen die Ablaufreihenfolge der Anweisungen fest. So verändert die Gruppe der „bedingten Verzweigungen" je nach Eintreten bestimmter Bedingungen den normalen sequentiellen Befehlsfluß. Die „unbedingte Verzweigung" löst dagegen in jedem Fall eine Verzweigung im normalen Programmfluß aus.

Eine unbedingte Verzweigung sieht normalerweise so aus:

```
BRA NEULABEL
```

Hier setzt sich der Befehlsfluß nach Ausführung von BRA von der Speicherstelle NEULABEL aus fort. Wenn der Distanzwert in einem vorzeichenbehafteten Acht-Bit-Wort untergebracht werden kann, läßt sich die gesamte Anweisung in einem Computerwort codieren. Die Hälfte des Wortes enthält dann den Op-code für BRA (hex 60), während das andere Byte den Distanzwert mit Vorzeichen angibt. Wenn der Distanzwert zu groß wird oder die Verzweigungsadresse noch nicht bekannt ist (der Assembler kann sie nicht berechnen, da er nur einen Vorwärtsbezug erhalten hat), steht das Distanzbyte auf Null, während das nächste Wort den vollen Distanzwert im 16-Bit-Format mit Vorzeichen angibt. Wir sehen uns diesen Vorgang später genauer an.

Für die meisten Situationen reicht BRA völlig aus. Manche Situationen – besonders berechnete Verzweigungsadressen – brauchen jedoch einen flexibleren Befehl. Nehmen Sie an, Sie möchten auf eine Tabellenadresse verzweigen, deren Index sich in einem Register befindet (die Adressierungsart „indirekt mit Index und Distanzwert"). Da mit BRA diese Art der Adreßberechnung nicht möglich ist, muß hier der Sprungbefehl JMP verwandt werden, der die Verzweigungsadresse gegebenenfalls berechnen kann.

Sehen wir uns die verschiedenen Spielarten der unbedingten Verzweigungen einmal an dem Programmbeispiel (‚Viele Verzweigungen') genauer an.

„PC relativ"

Die Sprungbefehle dieses Listings zeigen die absolute Adressierung (vorwärts und rückwärts) und die indirekte Adressierung mit Index und Distanzwert. Weitere Adressiermethoden sind „einfach indirekt" – zum Beispiel (A2) – und „PC relativ".

Die BRA-Befehle zeigen zunächst einen Distanzwert, der im Befehlswort enthalten ist, und dann zwei Anweisungen mit vollen Worterweiterungen (obwohl für den Distanzwert auch ein Byte genügt hätte). Da dem Assembler die Adresse von FINISH noch nicht bekannt war, mußte er für den Distanzwert eine volle Erweiterung einsetzen. Wenn Sie wissen, daß dieser Wert in ein vorzeichenbehaftetes Byte paßt, können Sie den Assembler mit dem Zusatz .S zwingen, die Kurzform zu nehmen. Unser Beispiel ließe sich daher auch kürzer als BRA.S FINISH schreiben.

Die bedingten Verzweigungen lassen sich in drei Hauptgruppen unterteilen:

- Verzweigungen im Zweierkomplement
- Verzweigungen ohne Vorzeichen
- Schleifensteuerung

Viele Verzweigungen

```
                    WAYOFF  EQU     $3FFF
                            ORG     $1000
100     3200        START   MOVE    D0,D1
1002    4EF8 1000           JMP     START     * Volle Worterweiterung
1006    4EF8 101C           JMP     FINISH    * noch ein Sprung nach vorn
100A    4EE8 0005           JMP     5(A0)     * indirekt mit Adreßdistanzwert
100E    4EF2 000C           JMP     12(A2,D0) * indirekt mit Index und
                                                Adreßdistanzwert

  * Spielarten der unbedingten Verzweigung
  * Wenn der Adreßdistanzwert aus einem Byte besteht, kann
    er im Op-Code-Wort untergebracht werden.

1012    60EC                BRA START         * Negativer Adreßdistanzwert
1014    6000 0004           BRA FINISH        * Vorwärtssprung – genaue
                                                Adresse noch unbekannt
1018    6000 2FE5           BRA WAYOFF        * Computerwort mit einem
                                                Distanzwert im 16-Bit-Format
101C    3200        FINISH  MOVE D0,D1
```

Das Befehlsformat der Gruppen ist identisch:

```
Bcc LABEL
```

cc bezieht sich auf die getesteten Bedingungscodes. Ist die Bedingung wahr, dann findet die Verzweigung auf LABEL statt, falls nicht, wird dann lediglich der nächste sequentielle Befehl angesprochen.

Die Spalte „Wahr, wenn –" zeigt die arithmetische Bedingung an, die als Ergebnis der Vergleichsbefehle CMP oder SUB entsteht. Bei der ersten Bedingungsgruppe der Tabelle (Verzweigungen im Zweierkomplement) wird das Bit V (Überlauf) in das logische Testen mit eingeschlossen. Dieses Merkmal bestimmt die Gruppenzugehörigkeit dieser Befehle und zeigt außerdem an, daß für die korrekte Ausführung auch das Überlaufbit getestet werden muß. Die genauen logischen Bedingungen der Vezweigungen sollten Sie dem Anwenderhandbuch von Motorola entnehmen. Eine BGE-Verzweigung testet beispielsweise, ob N=V. Der Sprung wird ausgelöst, wenn NOT N AND NOT V wahr ist (d. h. es existiert kein Überlauf, und die Zahl ist nicht negativ) oder N und V sind wahr (Überlauf und negative Zahl).

Ein Beispiel: Wir wollen die beiden vorzeichenbehafteten Zahlen D1 und D2 miteinander vergleichen und auf GROESSER verzweigen, wenn D2 größer ist als D1. Die Bedingung wird dabei mit einem Vergleichsbefehl getestet, auf

den die bedingte Verzweigung BGT folgt:

```
CMP D1,D2          * ergibt D2 — D1
BGT D2GROESSER     * verzweige auf
                     D2GROESSER,
                   * wenn nicht Null
                     und nicht
                   * negativ und kein
                     Überlauf
```

Die zweite Gruppe der bedingten Verzweigungen (in der Tabelle) arbeitet mit Zahlen ohne Vorzeichen und testet die Bedingungscodes auf einfachere Weise. So kann beispielsweise der Vergleich des Inhalts der Speicherstelle BETTY mit D1 unabhängig von der Überlaufbedingung ausgeführt werden:

```
CMP BETTY,D1    * ergibt D1 — BETTY
BEQ GLEICH      * verzweige auf GLEICH
                  wenn Z=1
```

Assemblerversion

Weitere Beispiele der bedingten Verzweigung ergeben sich aus der Assemblercodierung von Schleifen. So sieht der Ablauf

```
FOR I := 1 TO 5 DO
  (Programmteil fünfmal ausführen)
NEXT I
```

in der Assemblerversion so aus:

```
   MOVEQ #5,D7   * Schleifenzähler setzen
LOOP (Programmteil fünfmal ausführen)
   SUBQ #1,D7    * Zähler dekrementieren
   BNE LOOP      * Wiederholen bis
                   D7 = 0
```

Mit „Quick"-Befehlen läßt sich die Schleife in nur drei Worten codieren.

Die dritte Gruppe der bedingten Verzweigung bildet der Befehl DBcc, der dekrementiert und bei Eintritt der Bedingung auf cc verzweigt. Dieser Befehl ist eine Erweiterung des oben angeführten Schleifensteuermoduls, enthält in seinem Code aber bereits Dekrementierung und bedingte Verzweigung. DBcc ähnelt dem PASCAL-ähnlichen Pseudo Code für eine REPEAT-Schleife:

```
REPEAT
  (. . . Schleifeninhalt . . .)
UNTIL
  'C' = wahr oder Dn = —1
```

cc ist eine der Bedingungen (im zweiten Teil der Tabelle dargestellt) und Dn das Datenregister mit dem Schleifenzähler. Das vorige Beispiel wird mit DBcc so codiert:

```
   MOVEQ #5,D1   * Schleifenzähler setzen
LOOP (Programmteil fünfmal ausführen)
   DEBQ D1,LOOP  * Ende, wenn das zuletzt
                 * getestete 'cc' 0 ist
                 * oder D1=1
                   (6Durchläufe)
```

Beachten Sie den Unterschied in den Endbedingungen und auch die Unterschiede zu normalen BEQ-Befehlen. Wenn kein bedingtes Testen nötig ist, dann läßt sich DBcc mit einem auf F (falsch) gesetzten cc für einfache FOR-Schleifen verwenden, beispielsweise:

```
   MOVEQ #4,D3
LOOP (. . . Inhalt der FOR-Schleife . . .)
   DBF D3,LOOP
```

Diese Version entspricht der ursprünglichen FOR-Schleife mit fünf Wiederholungen und belegt auch die gleiche Menge Speicher.

Verzweigungsbedingungen

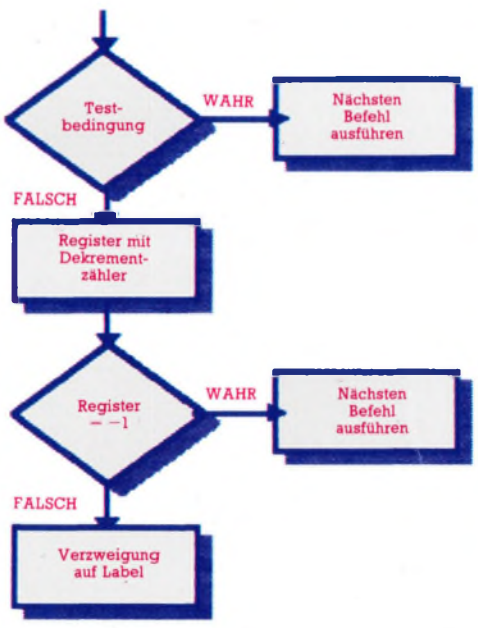
Vor der Verzweigung auf eine Adresse, die mit der Arithmetik im Zweierkomplement berechnet wird, werden die folgenden Bedingungen getestet:

cc Bedingung	Wahr, wenn
GT größer als	Ziel > Quelle
LT kleiner als	Ziel < Quelle
GE größer als oder gleich	Ziel >= Quelle
LE kleiner als oder gleich	Ziel <= Quelle
VS Überlauf	V = 1
VC kein Überlauf	V = 0

Vor der Verzweigung auf eine Adresse, die mit der Ganzzahlenarithmetik ohne Vorzeichen berechnet wird, werden die folgenden Bedingungen getestet:

EQ gleich Null	Z = 1
NE ungleich Null	Z = 0
MI minus	N = 1
PL plus	N = 0
HI höher als	C = Z = 0
LS niedriger als oder gleich	C oder Z = 1
CS Übertrag gesetzt	C = 1
CC Übertrag gelöscht	C = 0

Unter einer Bedingung



Der Befehl DBcc bietet dem Programmierer des 68000 eine Fülle von Möglichkeiten auf hoher Ebene. Das Ablaufdiagramm zeigt die Ausführung des Befehls. Beachten Sie, daß die Anweisung nicht nur ein Zählerregister dekrementiert, sondern auch eine Bedingung testet. Für eine feste Zahl von Wiederholungen (vergleichbar mit der FOR. . .NEXT-Schleife in BASIC) können Sie das Testen der Bedingung mit dem bedingten FALSCH (DBF) abstellen.

. . . hat man Töne

Das Betriebssystem der Schneider-CPC-Reihe enthält Möglichkeiten zur Tonsteuerung, die den BASIC-Klangbefehlen nahekommen. Wir zeigen, wie das OS Klänge erzeugt und wie man Programme mit Hintergrundmusik untermalen kann.

Die Schneider-CPC-Reihe arbeitet mit dem Tongenerator AY-3-8912 von General Instruments (kurz „8912" genannt), der nicht nur Klänge erzeugen kann, sondern über einen Acht-Bit-Ausgang auch die Tastatur abfragt.

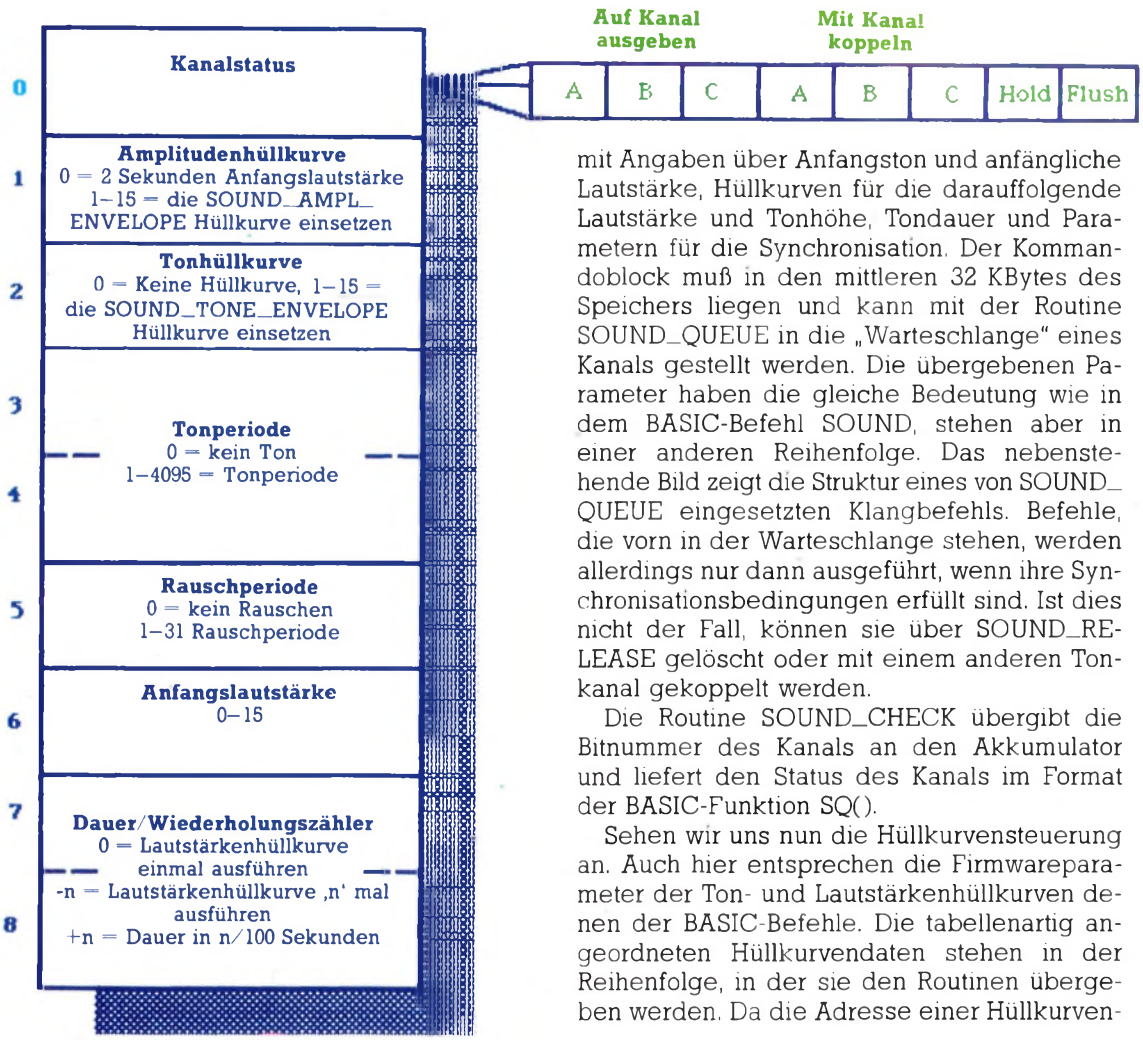
Der 8912 unterstützt drei unabhängige Tonkanäle (A, B und C) und einen Pseudogenerator für Rauschen, der auf jeden Kanal geschaltet werden kann. Der Chip erzeugt Töne, deren Lautstärke er mit vordefinierten Hüllkurven verändern kann. Die Aussage des Tongenerators wird zu Stereoklängen vermischt: Kanal A ist links, Kanal B rechts und Kanal C wird gleichmäßig zwischen beiden aufgeteilt. Diese Grundmöglichkeiten werden noch von dem Betriebssystem verfeinert, das per Software Ton, Lautstärke und Synchronisation steuert.

Bei der Tonerzeugung arbeiten die BASIC-Befehle ENV, ENT, RELEASE, SOUND und SQ direkt mit dem Betriebssystem, so daß Parameter und Konzepte fast völlig übereinstimmen. Das OS steuert Klänge mit einem speziellen Interrupt, der hundertmal pro Sekunde stattfindet und speziell auf die Verarbeitung von Tonbefehlen ausgerichtet ist.

Kommandoblöcke

Dabei werden beispielsweise Kanäle miteinander gekoppelt und Hüllkurven während des Ereignisses aktualisiert. Sehen wir uns zunächst die Klangbefehle und die Arbeitsabläufe genauer an.

Zu jedem Ton gehört ein Kommandoblock



mit Angaben über Anfangston und anfängliche Lautstärke, Hüllkurven für die darauffolgende Lautstärke und Tonhöhe, Tondauer und Parametern für die Synchronisation. Der Kommandoblock muß in den mittleren 32 KBytes des Speichers liegen und kann mit der Routine SOUND_QUEUE in die „Warteschlange" eines Kanals gestellt werden. Die übergebenen Parameter haben die gleiche Bedeutung wie in dem BASIC-Befehl SOUND, stehen aber in einer anderen Reihenfolge. Das nebenstehende Bild zeigt die Struktur eines von SOUND_QUEUE eingesetzten Klangbefehls. Befehle, die vorn in der Warteschlange stehen, werden allerdings nur dann ausgeführt, wenn ihre Synchronisationsbedingungen erfüllt sind. Ist dies nicht der Fall, können sie über SOUND_RELEASE gelöscht oder mit einem anderen Tonkanal gekoppelt werden.

Die Routine SOUND_CHECK übergibt die Bitnummer des Kanals an den Akkumulator und liefert den Status des Kanals im Format der BASIC-Funktion SQ().

Sehen wir uns nun die Hüllkurvensteuerung an. Auch hier entsprechen die Firmwareparameter der Ton- und Lautstärkenhüllkurven denen der BASIC-Befehle. Die tabellenartig angeordneten Hüllkurvendaten stehen in der Reihenfolge, in der sie den Routinen übergeben werden. Da die Adresse einer Hüllkurven-

tabelle jederzeit mit SOUND_T_ADDRESS oder SOUND_A_ADDRESS festgestellt werden kann, lassen sich die Hüllkurven leicht durch Patchen, ohne Aufruf der entsprechenden Firmwareroutinen, ändern.

Das Format dieses Datenblocks haben wir in dem nebenstehenden Diagramm abgebildet. Über den Offset in den Datenblock können Sie jeden beliebigen Bereich patchen. Sie müssen nur darauf achten, daß der Datenbereich während des Patchens nicht auch von den Firmwareroutinen angesprochen wird. Dies stört zwar den Programmablauf nicht, doch können dadurch Töne entstehen, die so nicht geplant waren.

Die Firmwareroutine SOUND_ARM_EVENT hat eine besonders praktische Funktion. Sie legt ein Ereignis an, das ausgelöst wird, wenn die Warteschlange eines angegebenen Kanals leer ist. Die Routine meldet dafür beim Klanginterrupt ein spezielles Ereignis an. Dieses Ereignis prüft zunächst, ob die Warteschlange des entsprechenden Kanals leer ist. Falls ja, wird der von SOUND_ARM_EVENT angelegte Ereignisblock aktiviert und je nach Ereignisklasse verarbeitet.

Der übergebene Ereignisblock wird mit KL_INIT_EVENT intitialisiert und kann jede Klasse oder Priorität annehmen. Da der Klanginterrupt aber alle Hundertstelsekunden stattfindet, genügt normales, asynchrones Ereignis.

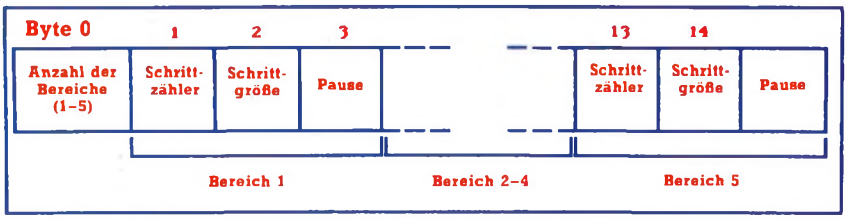
Mit dieser Technik läßt sich beim Ablauf eines Programms eine permanente Hintergrundmusik erzeugen, ohne daß der Tonchip ständig mit neuen Daten versorgt werden muß. Die Ereignisroutine holt sich dabei die nächste Tonprogrammadresse aus dem vordefinierten Datenbereich und ruft dann SOUND_QUEUE auf. Wenn Sie am Ende der Routine wieder an den Anfang verzweigen, erhalten Sie eine ständige Hintergrundbegleitung.

Klangereignis aus

Da SOUND_QUEUE das Klangereignis abschaltet, muß sich die Routine vor ihrem Ende selbst wieder aufrufen, indem sie mit SOUND_ARM_EVENT ihren eigenen Parameterblock übergibt. Das Ablaufdiagramm zeigt, wie die Ereignisroutine mit Hilfe der Warteschlange eine permanente Klangkulisse erzeugt.

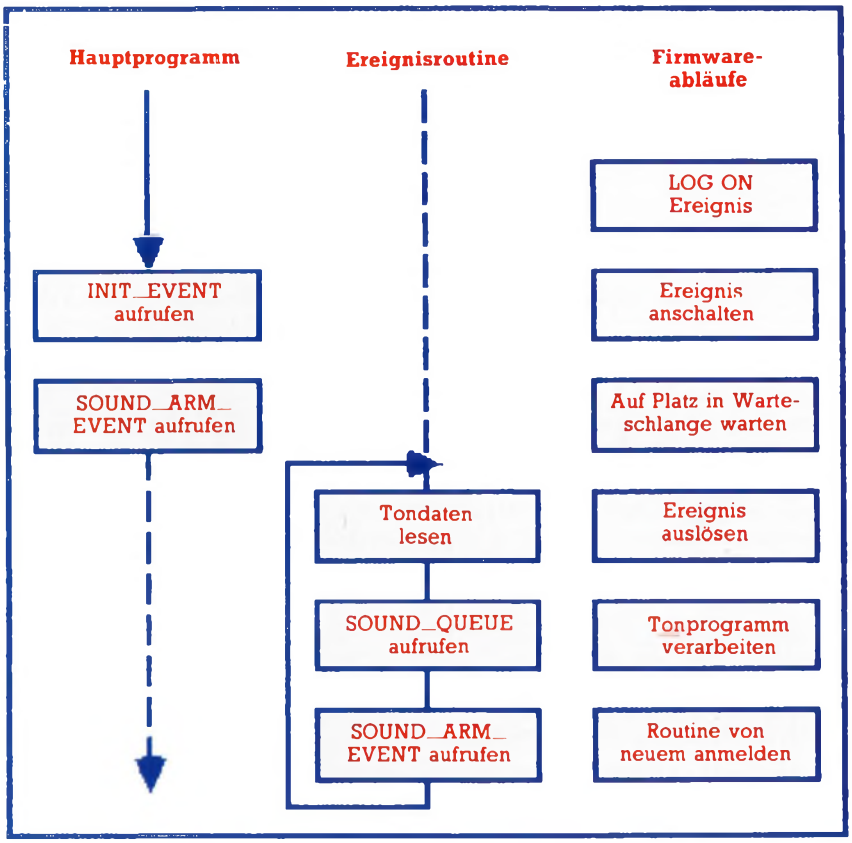
SOUND_HOLD kann die Klangerzeugung jederzeit abstellen. Dabei werden alle aktiven Hüllkurven angehalten und der Ton abgeschaltet. SOUND_CONTINUE, SOUND_QUEUE oder SOUND_RELEASE starten die Musik von neuem. SOUND_RELEASE löst den Start der Klangprogramme aus, die einzeln aufgerufen werden müssen.

Wenn mit der Tastatur eine Unterbrechung veranlaßt wurde, stellt der Aufruf von SOUND_HOLD sicher, daß fehlerhafte Töne nicht ausgeführt werden. So kann kein Mißklang oder eine Disharmonie entstehen.



Jede Hüllkurve kann bis zu fünf Bereiche enthalten. Jeder Bereich besteht aus drei Bytes, die als Schrittzähler dienen und Schrittgröße und Unterbrechungszeiten angeben. Wenn die Lautstärkenhüllkurven der Firmware eingesetzt werden, steht im ersten Byte des Blocks die Hüllkurvennummer (zwischen 128 und 143), und Byte zwei und drei enthalten die Unterbrechungszeit.

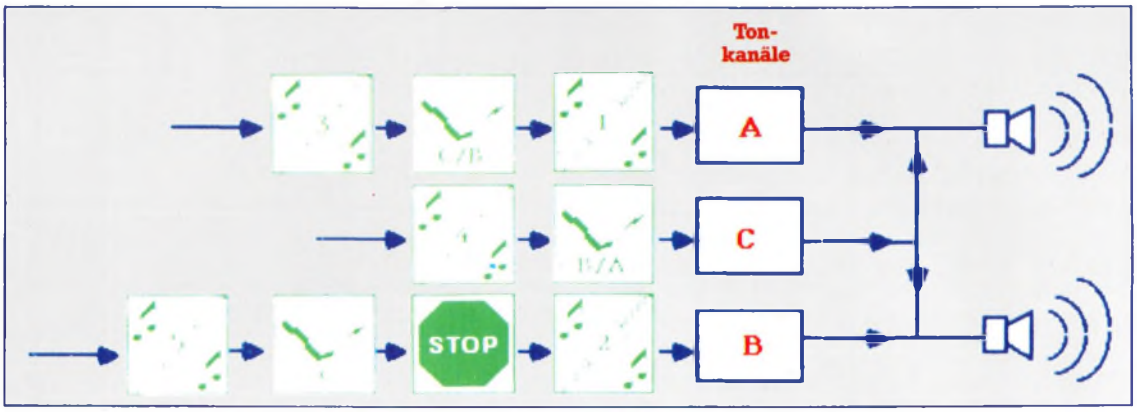
Mit SOUND_ARM_EVENT können Sie ereignisgesteuerte Hintergrundmusik erzeugen, während das Hauptprogramm im Vordergrund weiterläuft. Das Bild zeigt die verschiedenen Abläufe.



Der Routinen der Tonsteuerung

OBCA7H SOUND_RESET	RESET der Tonsteuerung
OBCAAH SOUND_QUEUE	Ton in eine der Warteschlangen stellen
OBCADH SOUND_CHECK	Liefert den Status einer Warteschlange
OBCAOH SOUND_ARM_EVENT	Ermöglicht das Auslösen eines Ereignisses, wenn die Warteschlange leer ist
OBCB3H SOUND_RELEASE	Gibt alle Töne einer Warteschlange frei
OBCB6H SOUND_HOLD	Hält Töne aller Warteschlangen an
OBCB9H SOUND_CONTINUE	Läßt gestoppte Töne weiterlaufen
OBCBCH SOUND_AMPL_ENVELOPE	Initialisiert eine der softwaregesteuerten Hüllkurven für die Lautstärke
OBCBFH SOUND_TONE_ENVELOPE	Initialisiert eine der softwaregesteuerten Hüllkurven für Töne
OBCC2H SOUND_A_ADDRESS	Liefert die Adresse einer Lautstärkenhüllkurve
OBCC5H SOUND_T_ADDRESS	Liefert die Adresse einer Tonhüllkurve

In der hier dargestellten Situation werden gerade die Klänge Eins und Zwei ausgeführt. Nach dem Ende von Klang Zwei wartet der nächste Befehl der Warteschlange auf SOUND_RELEASE. Da der Befehl in Warteschlange C mit den Kanälen A und B gekoppelt werden soll, muß er auf die Befehle dieser Warteschlange warten. Inzwischen ist Klang Eins beendet, so daß A mit C gekoppelt werden kann, doch fehlt noch die Kopplung mit B. Nach Freigabe der Warteschlange B ist die Kopplung möglich, und Klang Drei, Vier und Fünf ertönen gleichzeitig.



Symbole:

Tonbefehl (mit Nummer)

Kopplung mit dem angezeigten Kanal

Warten, bis die Freigabe von SOUND_RELEASE erfolgt

Der Befehl CALL

Der BASIC-Befehl CALL, mit dem Daten als „Parameterblock" an Maschinencoderoutinen übergeben werden, ist in den Handbüchern nicht besonders gut beschrieben. Die vom Anwender gelieferten Parameter werden von der untersten (ersten) Blockadresse an aufwärts abgelegt. Beim Einsprung in die mit CALL aufgerufene Routine zeigt IX auf diese Adresse.

Numerische Werte können direkt übergeben werden, nicht aber Variable. Statt dessen wird die Adresse der Variablen (bei Stringvariablen die Adresse des String-Steuerblocks) mit dem Symbol @ übergeben.

Das folgende Beispiel ruft eine Routine bei &A000 auf, übergibt ihr den Wert 85 und enthält bei der Rückkehr ins BASIC die Resultate in Ergebnis% und Antwort$:

```
CALL &A000,85,@Ergebnis%,@Antwort$
```

Beim Einsprung in die Routine zeigt IX auf einen ROM-Block, der zuerst den Wert 85 als 16-Bit-Ganzzahl ohne Vorzeichen enthält, dann zwei Bytes mit der Adresse von Ergebnis% und schließlich die Adresse des Stringsteuerblocks für Antwort$. Der Stringsteuerblock besteht aus drei Bytes, in denen zunächst die Stringlänge und dann die Adresse stehen. Dieser Block sollte möglichst nicht geändert werden, da sonst Probleme mit der Firmware entstehen könnten.

Das Editieren von Hüllkurven kann recht kompliziert werden. Mit unserem Editor für Hüllkurven können Sie Ihre Parameter eingeben und auf dem Bildschirm gleich die Form der dazugehörigen Hüllkurve sehen.

Editor für Hüllkurven

```
50 MEMORY &6FFF: DIM envelope(15)
60 done=0: routine=&7000: buffer=routine+&100: nk=1
70 WHILE -1
80  MODE 2: LOCATE 1,20: INPUT "Start volume (0-15) ";sv%: I
F sv%)15 OR sv%<0 THEN 80
90 LOCATE 1,21: INPUT "Start tone ";tt: IF tt<0 OR tt)4095 T
HEN 90
100 LOCATE 1,22:INPUT "Horizontal scale (1-10) ";sc%: IF sc%
)10 OR sc%<1 THEN 100
110 LOCATE 1,23:INPUT "Volume or tone envelope (V/T) ";senv$
: IF (UPPER$(senv$)<>"V") AND (UPPER$(senv$)<>"T") THEN GOTO
 110
120 VV=0: IF UPPER$(senv$)="V" THEN pk=&BC: VV=1:  ELSE pk=&
BF
130  length%=0: soundx=0: IF VV=1 THEN soundy=INT(sv%*9.7) E
LSE soundy=68
140  FOR x=1 TO 15
150   envelope(x)=0
160  NEXT
170  GOSUB 270
180  LOCATE 1,20: PRINT CHR$(18)
190  FOR count=1 TO 5
200   GOSUB 360: nk=ABS(nk-1): GOSUB 540: IF VV=1 THEN GOSUB
 880 ELSE GOSUB 890
210  NEXT
220  WHILE NOT done
230   LOCATE 5,20: INPUT "Edit (y/n) ";ed$
240   IF UPPER$(ed$)="N" THEN done=-1 ELSE GOSUB 790
250  WEND
260 WEND
270 REM intialise screen
280 MODE 1: WINDOW #1,1,39,1,2: GOSUB 1000
290 MOVE 20,380: DRAW 20,210,3: DRAW 600,210,3
300 MOVE 21,soundy+211
310 POKE routine, &3E: POKE routine+1,1
320 POKE routine+2, &21: POKE routine+3, buffer-(INT(buffer/
256))*256: POKE routine+4, INT(buffer/256)
330 POKE routine+5,&C3: POKE routine+6,pk: POKE routine+7,&B
C
340 POKE routine+8,&C9
350 RETURN
360 REM get section data
370 LOCATE 1,20: PRINT CHR$(18);: LOCATE 5,20: PRINT "Sectio
n ";count
380 LOCATE 1,22: PRINT CHR$(18);: LOCATE 5,22: INPUT "Step c
ount "; skip%: IF VV=0 THEN GOTO 400 ELSE IF skip%<0 OR skip
%)127 THEN 380
390 GOTO 410
400 IF skip%<0 OR skip%)239 THEN GOTO 380
410 P$=STR$(SKIP%): GOSUB 900
420 LOCATE 1,23: PRINT CHR$(18);: LOCATE 5,23: INPUT "Step s
ize ";size%: IF VV=0 THEN GOTO 440 ELSE IF ABS(size%))15 THE
N 420
430 GOTO 450
440 IF ABS(size%))127 THEN 420
450 P$=STR$(size%): GOSUB 900
460 ss=size%: IF size%<0 THEN size%=256-ABS(size%)
470 LOCATE 1,24: PRINT CHR$(18);: LOCATE 5,24: INPUT "Pause
time "; pause%: IF pause%)255 THEN 470
480 P$=STR$(pause%): GOSUB 900
490 envelope ((count-1)*3+1)=skip%: envelope ((count-1)*3+2)
=size%: envelope ((count-1)*3+3)=pause%
500 FOR x=20 TO 24
510  LOCATE 1,x: PRINT CHR$(18);
520 NEXT
530 RETURN
540 REM envelope intialisation
550 POKE buffer,count
560 CLS #1: GOSUB 1000
570 FOR x=0 TO count-1
580 POKE buffer+(x*3+1),envelope(x*3+1): skip%=envelope(x*3+
1)
590 p$=STR$(envelope(x*3+1)): GOSUB 900
600 POKE buffer+(x*3+2),envelope(x*3+2): size%=envelope(x*3+
2)
610 pp=envelope(x*3+2): IF pp)127 THEN pp=pp-256
620 p$=STR$(pp): GOSUB 900
630 POKE buffer+(x*3+3),envelope(x*3+3): pause%=envelope(x*3
+3)
640 p$=STR$(envelope(x*3+3)): GOSUB 900
650 NEXT x
660 length%=length%+skip%*pause%
670 CALL routine
680 IF VV=0 THEN GOSUB 910: RETURN
690 off%=size%*10: IF size%)127 THEN off%=(size%-256)*10
700 FOR x=1 TO skip%
710  soundy=(soundy+off%) MOD 150: IF soundy<0 THEN soundy=1
50-soundy
720   FOR c=1 TO (10/51)*pause%*2: soundx=soundx+(sc%)
730    DRAW soundx*21,soundy+211,2+nk
740   NEXT
750 NEXT
760 DRAW soundx*21,soundy+211,2+nk
770 RETURN
780 REM redraw graph
790 LOCATE 5,20: PRINT CHR$(18);: INPUT "Section";sec%: IF s
ec%<1 OR sec%)5 THEN 790
800 count=sec%: GOSUB 360
810 CLS: soundx=0: soundy=INT(sv%*9.7): length%=0: GOSUB 270
820 FOR section=1 TO 5
830   count=section: GOSUB 540
840 NEXT
850 GOSUB 870
860 RETURN
870 REM sound
880 SOUND 1,tt,length%,sv%,1: RETURN
890 SOUND 1,tt,length%,sv%,,,1: RETURN
900 PRINT #1,MID$(P$,(2-ABS(VAL(p$)<0)));",";: RETURN
910 FOR x=1 TO skip%
920  soundy=soundy-((15/127)*ss): IF soundy<0 THEN soundy=0
930 IF soundy)189 THEN soundy=189
940  FOR c=1 TO (7/51)*pause%*2: soundx=soundx+sc%
950   DRAW soundx*21,soundy+211,2+nk
960  NEXT
970 NEXT
980 DRAW soundx*21,soundy+211,2+nk
990 RETURN
1000 IF VV=1 THEN PRINT #1, "ENV 1, "; ELSE PRINT #1, "ENT 1
, ";
1010 RETURN
```

Sprunghafte Routine

Programme, die mit Subroutinen arbeiten, sind flexibel, lassen sich leicht ändern und machen die Fehlersuche einfach. In dieser Folge untersuchen wir die JSR-Anweisung des 68000 und zeigen, wie sie eingesetzt wird.

Wir wollen den Befehl JSR (Sprung zum Unterprogramm) zunächst an einem einfachen Beispiel erläutern. Nehmen wir ein Programm, das ein Array von zehn Wörtern nach der Formel

```
TARRAY[I] := ARRAY[I] ^2 + 20*ARRAY[I]
```

umwandelt und die Summe aller Elemente von TARRAY in die Speicherstelle SUM stellt. Die Formel sieht zwar kompliziert aus, besagt aber nur: „Quadriere jedes Arrayelement einzeln, addiere darauf das Zwanzigfache des Elements und speichere das Ergebnis in dem entsprechenden Element von TARRAY."

Das erste Programmlisting zeigt, wie der 68000 die Umwandlung erledigt. Um feststellen zu können, ob das Programm richtig arbeitet, laden wir die Testdaten (1 bis 10) in ARRAY und prüfen TARRAY. Nach Ablauf des Programms müssen die Werte von TARRAY 21, 44,69,96 etc. betragen. Beachten Sie, daß nicht festgelegt wurde, für welchen Zahlenbereich ARRAY gültig ist. Bei einem fehlerlosen Programm sollten die Umwandlungsergebnisse und die Summe in eine Wortlänge (16 Bits) passen.

Unser Beispiel ist natürlich nicht der einzige Weg, diese Aufgabe zu lösen. Hier kommen Programmstruktur und die Bedeutung der Subroutine, speziell in der Assemblersprache, ins Spiel.

Zwar gibt es viele Möglichkeiten, Programme zu strukturieren, doch arbeiten fast alle „sauberen Programme" (d. h. Programme, die Sie auch später noch verstehen) mit der „funktionellen Aufsplittung". Dabei wird ein Programm in Bereiche aufgeteilt, die bestimmte Funktionen ausführen. Die Funktionen lassen sich mit Begriffen wie „Berechnen", „Umstellen" oder „Prüfen" beschreiben. In Hochsprachen heißen diese Codebereiche „Prozeduren" und die dorthin übergebenen Daten „Parameter".

Der Assemblercodierung steht nur die modulare Struktur der Subroutine zur Verfügung. Auf dem 68000 wird sie so aufgerufen:

```
JSR SUBR
```

Dabei ändert sich der Programmfluß durch einen Sprung auf den Codebereich mit dem Label SUBR. Dieser Code wird ausgeführt, bis er auf den Befehl RTS trifft. An dieser Stelle verzweigt der Programmfluß zu der Anweisung, die dem JSR-Befehl folgt (RTS bedeutet „Rückkehr aus dem Unterprogramm").

Doch zurück zu unserem Programmbeispiel. Statt „in einem Stück" zu quadrieren, multiplizieren und addieren, läßt sich dieser Vorgang auch abkoppeln und in einer Subroutine unterbringen. Das Programmlisting zwei zeigt den neuen Aufbau. Dabei steht in Zeile 6 der Aufruf der Subroutine CALC, und die früheren Zeilen 6 bis 9 befinden sich in Zeile 12 bis 15 – mit einem RTS am Ende.

Beliebig oft CALC

Das bringt den Vorteil, daß ein klar abgegrenzter Codebereich (Zeile 12 bis 16) einen exakt definierten Ablauf mit den Daten ausführt, die über den Parameter D1 übergeben wurden. Sie haben nicht nur einen übersichtlicheren Programmaufbau, sondern können die Subroutine CALC auch sooft und von wo aus Sie wollen aufrufen. Die Ergebnisse der in D1 übergebenen Daten werden in D2 zurückgegeben.

Beim Anlegen von Subroutinen sollten Sie, besonders bei Mehrplatzanwendungen, darauf achten, daß nur ein Einsprungspunkt und ein Rücksprung existiert. Auch sollten Sie vermeiden, von der Subroutine aus das Hauptmodul aufzurufen und weiterhin nicht zu viele bedingte Rücksprünge anlegen, da dies die Fehlersuche sehr erschwert.

Subroutinen fördern nicht nur einen guten Programmaufbau, sondern bieten auch die Möglichkeit, Speicherplatz zu sparen. (So muß der Code von CALC nicht für jeden Einsatz wiederholt werden.)

Wenn Sie CALC an mehreren Punkten des Programms aufrufen, brauchen Sie dann bei einer Änderung (beispielsweise der Umwandlung) lediglich einen Programmbereich zu ändern.

Nehmen wir an, die Umformung sollte

```
D2 := D1^2-20*D1
```

Programmbeispiel Eins

```
* Umformung von ARRAY in TARRAY mit der Formel
* TARRAY[I] := ARRAY[I]^2 + 20*ARRAY[I]
* Beide Arrays enthalten zehn Elemente
* Folgende Register werden eingesetzt:
*    AO zeigt auf ARRAY
*    A1 zeigt auf TARRAY
*    DO ist ein Schleifenzähler
*    D1 ist ein Zwischenspeicher
*    D2 ist eine Kopie von ARRAY[I]

* Schleifenzähler initialisieren
* den ersten Pointer setzen
* den zweiten Pointer setzen
* SUM auf Null stellen
* ARRAY[I] holen
* und kopieren
* Quadrat von ARRAY[I] bilden
* D1 wird das 20fache von
  ARRAY[I]
* Komponenten addieren
* in TARRAY[I] speichern
* Gesamtsumme aktualisieren
* Schleifenzähler dekrementieren
* wiederholen, falls nicht fertig
* Ende der Routine, zurück zum
  Monitor
```

Anmerkungen zum Programm:
Die Zeilen 1 bis 4 initialisieren die Programmvariablen. In Zeile 5 werden die Elemente von ARRAY mit indirekter Adressierung und Nach-Inkrementierung in D1 geladen. Zeile 6 legt eine Kopie in D2 an, damit ARRAY nicht nochmals angesprochen werden muß. Die Zeilen 7 und 8 bilden mit dem vorzeichenbehafteten Multiplikationsbefehl MULS die beiden Umformungskomponenten D1 und D2, die dann von Zeile 9 addiert werden.

Zeile 10 legt das Ergebnis in dem entsprechenden Element von TARRAY ab, und Zeile 11 addiert das Ergebnis zu der Gesamtsumme SUM. Zeile 12 und 13 steuern die Schleife, damit nur zehn Elemente umgewandelt werden. Das Programm ist eigentlich eine FOR... NEXT-Schleife zwischen den Zeilen 5 und 12/13, deren Schleifenzähler in Zeile 1 gesetzt wird.

Programmbeispiel Zwei

```
* Subroutine zur Berechnung
  aufrufen
* neuer Arraywert

* Subroutine zur Berechnung
  von D1^2+20*D1
* und Rücksprung mit dem
  Ergebnis in D2
* Rücksprung zum aufrufenden
  Programm
```

lauten, dann genügt in der neuen CALC-Routine nur die Änderung der Zeile 15 in

```
SUB D1,D2
```

Die Programme werden dadurch nicht nur flexibler, sondern lassen sich auch leichter von Fehlern befreien. So könnten Sie CALC beispielsweise mit jeder Art von Werten (in D1 übergeben) testen, die (bei Rückgabe in D2) keine Probleme im Hauptprogramm verursachen. Die Subroutine hat nur einen Einsprungspunkt (bei dem Adreßlabel CALC) und eine Rücksprunganweisung (RTS).

Sie sehen, daß die Kriterien Flexibilität, Anpassungsfähigkeit und Zuverlässigkeit erfüllt sind. Subroutinen haben jedoch auch Nachteile. Wir müssen uns dazu (und auch für den Mechanismus der Parameterübergabe) die Stackvorgänge des 68000 genauer ansehen.

„Nebenwirkungen"

Bei der Ausführung von JSR muß die Adresse des Befehls direkt hinter JSR (die Rücksprungverbindung) gespeichert werden. JSR schiebt diese Adresse daher (als die vier Bytes einer Langwortadresse) auf den Stack, während der Programmzähler (PC) mit der Adresse der Subroutine geladen wird. Wenn der Ablauf der Subroutine nun auf ein RTS trifft, wird die Rücksprungadresse wieder vom Stack gezogen und der PC damit geladen. Auf diese Weise geht die Programmausführung nach Beendigung der Subroutine weiter. Von all diesen Vorgängen merkt der Anwender nichts, da der 68000 sie automatisch ausführt. Achten Sie auf folgende „Nebenwirkungen".

Zunächst müssen Sie sicherstellen, daß der Stackpointer, SP (auf dem 68000 das Register A7), richtig gesetzt ist, da man sonst Programmcode oder Daten überschreibt oder gar Hardwareadressen ändert. Dies geschieht normalerweise mit:

```
STACK EQU $1000
* Stack auf 1000 hex setzen
BEGIN LEA STACK,SP
* Stackpointer initialisieren
```

Der Stack „wächst" nun von 1000 aus in Richtung Null (jeder ‚PUSH' ist demnach eine Vor-Dekrementierung).

Sie müssen aber auch darauf achten, daß der Stack nicht zu groß wird. Hier können Sie nur schätzen, wieviel Stackkapazität nötig ist – und dies ist manchmal sehr schwierig. Bei kurzen Programmen mit einfachen Subroutinenaufrufen ergibt eine Untersuchung des Codes und ein zusätzlicher Sicherheitsfaktor oft schon die richtigen Werte. Bei verschachtelten Subroutinen oder Rekursionen kann die Berechnung der Stackgröße so schwierig werden, daß Sie sich nur durch Ausprobieren an die richtigen Werte herantasten können.

In der nächsten Folge geben wir weitere Beispiele für Subroutinenaufrufe und konzentrieren uns besonders auf die Datenübergabe an und von Subroutinen. Hier bietet der 68000 einige interessante Befehle.

Hochstapelei

In den vorigen Folgen dieser Serie hatten wir festgestellt, daß sich Subroutinen ausgezeichnet zur Programmstrukturierung eignen. Wir wollen uns nun die Beziehung der Subroutinen zum Stapel ansehen und dann auf die Parameterübergabe eingehen.

Sehen wir uns zunächst ein Programmbeispiel an, das die Subroutine CALC zweimal aufruft:

```
                (. . . Codezeilen . . .)
2000 4EB8       JSR CALC      * Subroutine
                              aufrufen
                (. . . Codezeilen . . .)
2100 4EB8       JSR CALC      * Subroutine
                              nochmals
                              aufrufen
2102 4000       (. . . Codezeilen . . .)
4000 3200 CALC  MOVE D0,D7 * Subrouti-
                              nenein-
                              sprung
                (. . . Codezeilen . . .)
4100 4E75       RTS           * Rück-
                              sprung
```

Bei jeder Ausführung von JSR schiebt der 68000 den Inhalt des Programmzählers (die Adresse des auf JSR folgenden Befehls) auf den Stack und lädt die von JSR angegebene Adresse in den PC. Wenn der 68000 am Ende der Subroutine das RTS findet, zieht er die Rücksprungadresse (auch „Linkadresse" genannt) wieder vom Stack und lädt sie in den PC. Die Linkadresse wird im Langwortformat auf den Stack geschoben. In unserem Beispiel befinden sich – nach Ausführung des ersten CALC – die im nebenstehenden Bild gezeigten Werte auf dem Stack und im PC.

Die zweite Hälfte des Bildes zeigt, wie der Stack nach Ausführung von RTS und dem Laden der Linkadresse in dem PC aussieht. Die Linkadresse wird dabei auf dem Stack nicht gelöscht, sondern beim nächsten Stackeinsatz einfach überschrieben.

Die für den Linkvorgang nötigen Abläufe erledigt die Hardware des 68000.

Bei verschachtelten Subroutinen wird zur Speicherung der Rücksprungadressen der Stack erweitert. Wenn sich CALC jedoch selbst aufruft (Rekursion), werden die Rücksprungadressen solange „gestapelt", bis die Rekursion endet oder das Programm wegen des Stacküberlaufs abstürzt.

Sehen wir uns nun an, wie Ein- und Ausgabeparameter übergeben werden. In der letzten Folge hatten wir CALC über D1 mit Daten versorgt und das Ergebnis in D2 erhalten. Da die Registerzahl begrenzt ist, reicht diese Lösung jedoch nur für kleine Programme.

Bei längeren Programmen bietet sich, besonders bei der Arbeit mit Compilern, der Stack als Übergabemedium an. Die Parameterübergabe an CALC würde dann so aussehen:

```
2000 MOVE PARAM1,—(SP) * ersten Para-
                         meter auf den
                         Stack schieben
2004 MOVE PARAM2,—(SP) * zweiter Param.
2008 JSR CALC          * CALC aufrufen
200A . . .
```

Nun enthält der Stack beim Einsprung in CALC

```
                PARAM1
                PARAM2
                ls link
SP in CALC —>   ms link
```

Der Zugriff auf PARAM1 geschieht mit einem Distanzwert auf SP, damit die Linkadresse übersprungen wird:

```
ADDQ #6,SP      * Pointer korrigieren
MOVE (SP),D4    * Parameter holen
```

oder einfacher mit

```
MOVE 6(SP),D4
```

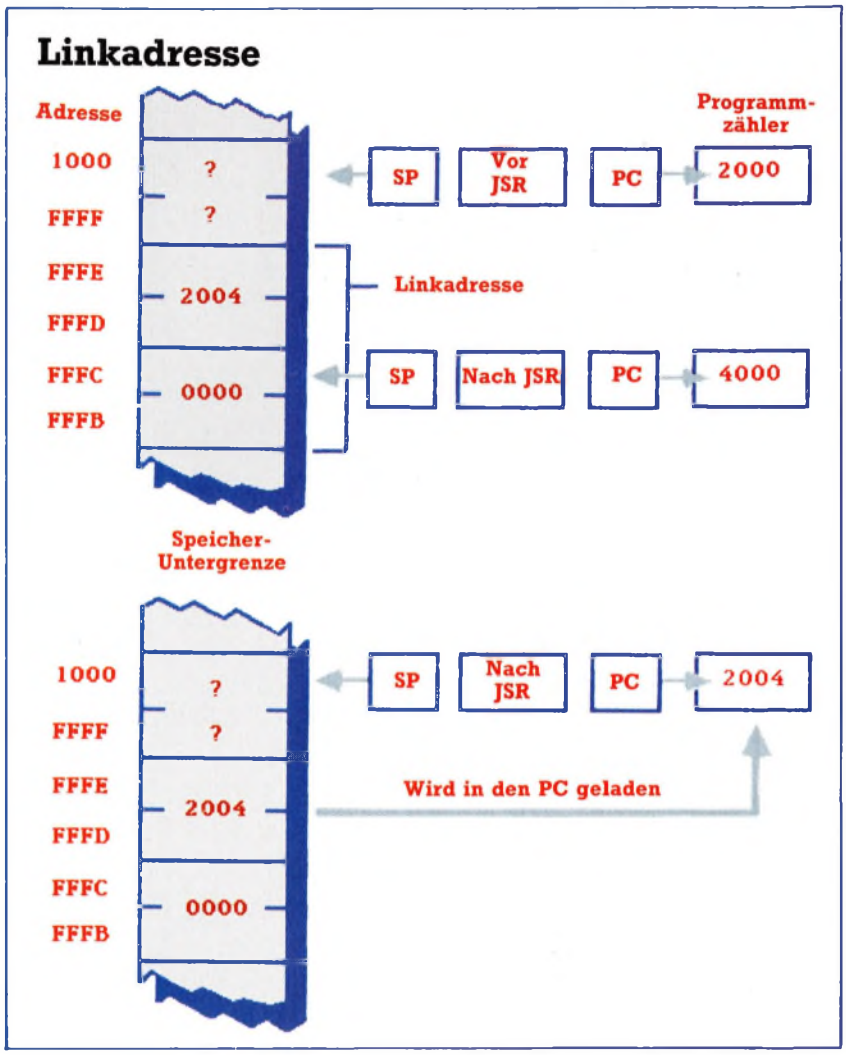
Bei einer Korrektur des Stackpointers ist sicherzustellen, daß SP vor Ausführung der Rücksprungadresse auf die Linkadresse zeigt, da Sie sonst unvorhersehbare Ergebnisse erhalten. Dies ist auch bei der Rückgabe der Ergebnisse per Stack wichtig, es sei denn, die Subroutine überschreibt einen oder beide Eingabeparameter.

Weit einfacher ist der Einsatz eines separaten Parameterstacks (beispielsweise A6), da dabei der Stackpointer nicht beeinflußt wird und für Hardwarefunktionen reserviert bleiben kann. In diesem Fall sieht der Aufruf so aus:

```
MOVE PARAM1,—(A6)  * ersten Parameter
                     auf den Stack A6
                     schieben
MOVE PARAM2,—(A6)  * zweiten Parameter
JSR  CALC          * SP für die Link-
                     adresse freihalten
```

In der Subroutine lassen sich die Parameter dann leicht mit

```
MOVE (A6)+,D2 * zweiten Parameter ho-
                len
MOVE (A6)+,D1 * ersten Parameter
                holen
```



Der 68000 legt eine Rücksprungadresse (die „Linkadresse") auf dem Stack ab, bevor er die Programmsteuerung an eine Subroutine übergibt. Diese Adresse wird im Langwortformat auf dem Stack abgelegt und beim Auftreten eines RTS-Befehls wieder in den Programmzähler eingesetzt.

abrufen, da keine Linkadressen übersprungen werden müssen. Beachten Sie, daß die Parameter in umgekehrter Reihenfolge eingelesen werden.

Stacks werden viel in modernen Hochsprachen mit Blockstruktur eingesetzt. Dabei ordnet die Sprache dem Parameter und den lokalen Variablen von Prozeduren Platz in einem fest definierten Speicherbereich zu. Hier ein Beispiel:

```
Procedure Berechnen (xval, yval : int);
  var
    store : int;
  begin
    store := 2*xval + yval^2;
```

Dieses PASCAL-Modul arbeitet mit den Parametern „xval" und „yval" und der lokalen Variablen „store", denen im Stack Platz zugeteilt wird. Alle weiteren, im Programm nicht aufgeführten Zwischenvariablen des Compilers (z. B. die yval^2-Komponente von „store" während der Bewertung von 2*xval) werden nach der „Push"- und „Pop"-Methode auf dem Stack abgelegt bzw. von dort abgerufen.

Der 68000 bietet für die Verwaltung der Stackdaten die beiden Befehle LINK (Adreßregister auf den Stapel schieben) und UNLK (Adreßverbindung lösen). Die Befehle werden gemeinsam eingesetzt und reservieren im Stack einen Speicherblock für den Subroutineneinsatz. Nach Einsprung in eine Subroutine richtet LINK das Adreßregister RZ („Rahmenzeiger") ein, das auf einen Datenbereich des Stacks zeigt, und bewegt den SP um einen bestimmten Distanzwert nach unten. Wenn ein Stack nach Ausführung von JSR so aussieht:

```
                PARAM1
                PARAM2
                ls link
        SP →    ms link
```

dann hat er nach LINK folgenden Inhalt:

```
                PARAM1
                PARAM2
                ls link
                ms link
        RZ →    alter RZ
Distanzwert . . . lokale Variable
        SP
```

Falls die Subroutine weiteren Speicher benötigt, wächst der Stack nach „unten".

Kurz vor Ende der Subroutine (d. h. vor RTS) versetzt UNLK die Pointer wieder in den Zustand vor dem Subroutinenaufruf.

Die beiden bisher erwähnten Fälle sind die zwei Pole der Parameterübergabe des 68000. Auf der einen Seite werden die Parameter einfach in Datenregister (bei der indirekten Parameterübergabe auch in Adreßregister) geladen, während andererseits der speziell strukturierte Stack die Parameterübergabe von Hochsprachen ermöglicht. Für den normalen Gebrauch reicht die Übergabe per Register jedoch oft nicht aus, während die Stacksteuerung zu kompliziert ist.

Hier schaffen andere Methoden Abhilfe. So wird zum Beispiel ein Datenbereich festgelegt, der die Ein- und Ausgabeparameter einer Gruppe von Subroutinen (d. h. eines Moduls) enthält. Dieser Datenbereich ist nicht global, sondern wird nur von dieser Subroutinengruppe für die Parameterübergabe eingesetzt. Dabei holt sich jede Subroutine ihre Variablen aus einem festen Speicherbereich und speichert dort auch die Ergebnisse. Hier ein Beispiel:

```
CALC  LEA    INPUTS,A4   * auf Eingabe-
                         parameter zei-
                         gen
      MOVE   (A4),D0     * ersten Para-
                         meter holen
      MOVE   (A4)+,D1    * zweiten Para-
                         meter holen
      . . . Programmcode
      MOVE   D5,OUTPUT   *Ausgabepara-
                         meter spei-
                         chern
      RTS
```

Wenn Sie den Pointer auf einen Parameterbereich in einem definierten Adreßregister über-

geben, wird die erste Zeile des Beispiels überflüssig. Diese Methode arbeitet ebenfalls mit separaten Stackbereichen für die Parameter INPUTS und OUTPUT.

Eine zweite Methode verwendet zwar Register, setzt damit aber den Befehl MOVEM ein. Sehen wir uns den Befehl einmal genauer an.

```
MOVEM Registerliste, SAVEAREA
```

legt die in der Registerliste angegebenen Register in einen Speicherbereich ab, der bei der absoluten Adresse SAVEAREA anfängt.

```
MOVE D3/D5/A2,SAVEAREA
```

lädt D3 in SAVEAREA, D5 in SAVEAREA+2 und A2 in SAVEAREA+4. MOVEM vereinfacht diesen Vorgang:

```
MOVEM D3/D5/A2,—(SP)
```

Der Befehl stapelt die Register in aufeinanderfolgenden, absteigenden Adressen. Beim Sichern einer Reihe von Registern erlaubt der 68000 auch folgende Kurzversion:

```
MOVEM D1—D5/A4,—(SP)
```

Hier werden D1 bis D5 und A4 gespeichert.

MOVEM kann die Register (als Zieloperand) aber auch zurückladen:

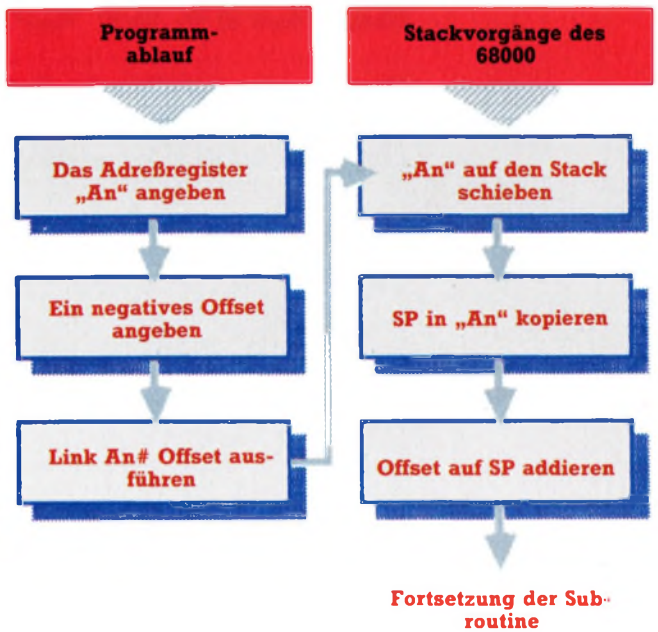
```
MOVEM—(SP),D1—D5/A4
```

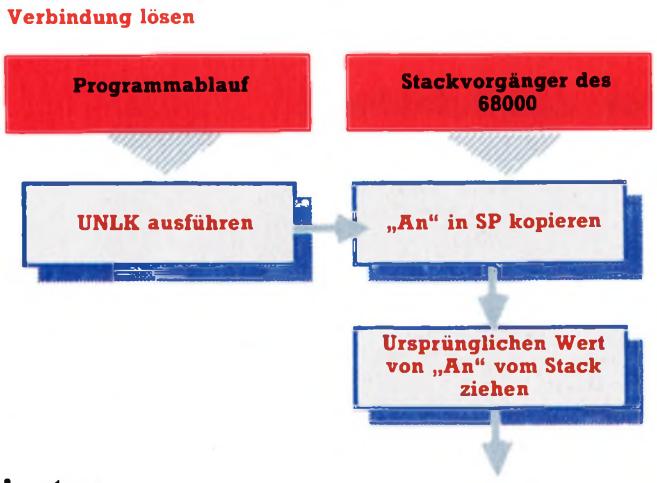
setzt die Register wieder auf ihre ursprünglichen Inhalte.

Bei der Parameterübergabe lassen sich mit dieser Methode leicht Register freisetzen, die sonst für spezielle Zwecke gebraucht werden (z. B. zur Systemsteuerung). Wenn beispielsweise D0 und D1 als Parameterregister definiert sind und (die sonst für Systemzwecke reservierten Register) D2 bis D7 für die Subroutine eingesetzt werden sollen, kann MOVEM sie für die Dauer der Subroutine freistellen. Z. B.:

```
CALC MOVEM  D2—D7,—(SP)  * alten Inhalt
                         von D2—D7
                         sichern
     MOVE   D0,D2        * ersten Pa-
                         rameter la-
                         den
     ...                 * Code, der
                         D2—D7 ein-
                         setzt
     MOVEM  —(SP),D2—D7  * alten Inhalt
                         zurückladen
     RTS
```

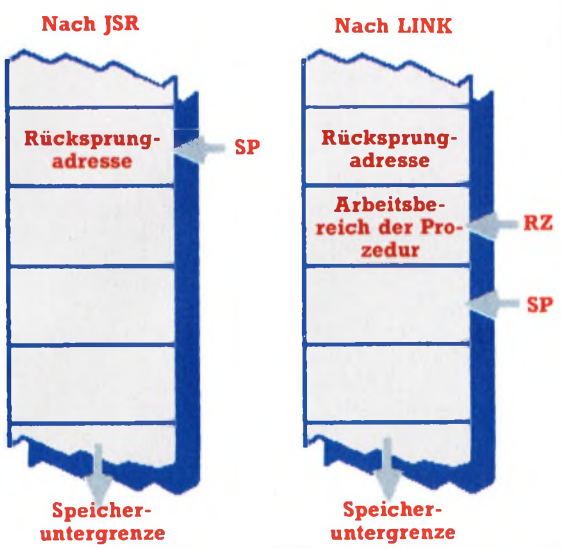
Sie sehen, der 68000 hat mehrere Möglichkeiten, Parameter an Subroutinen zu übergeben, und verfügt für diese Aufgabe auch über entsprechende Befehle.





Reservierte Plätze

Mit LINK und UNLK können Programme sehr schnell einen Stackbereich für Subroutinen reservieren. Dies ist besonders bei Mehrplatzanwendungen wichtig, da Subroutinen hier von anderen Programmen unterbrochen werden können. LINK bewegt den Stackpointer um einen Distanzwert abwärts. Das Register „An“ dient als „Rahmenzähler“, mit dem die Subroutine auf die Daten des reservierten Speicherblocks zugreift. UNLK stellt schließlich die ursprünglichen Werte des SP und von „An“ wieder her.



Bit für Bit

Auf 68000-Systemen benutzt die Peripherie den gleichen Kommunikationsbus, an den auch Speicher und CPU angeschlossen sind. Wir untersuchen die Ein- und Ausgabe (E/A) und wie die CPU mit der Methode der „Memory Map" arbeitet.

Es gibt zwei Hauptmethoden, E/A-Geräte anzuwählen und mit ihnen Daten auszutauschen:

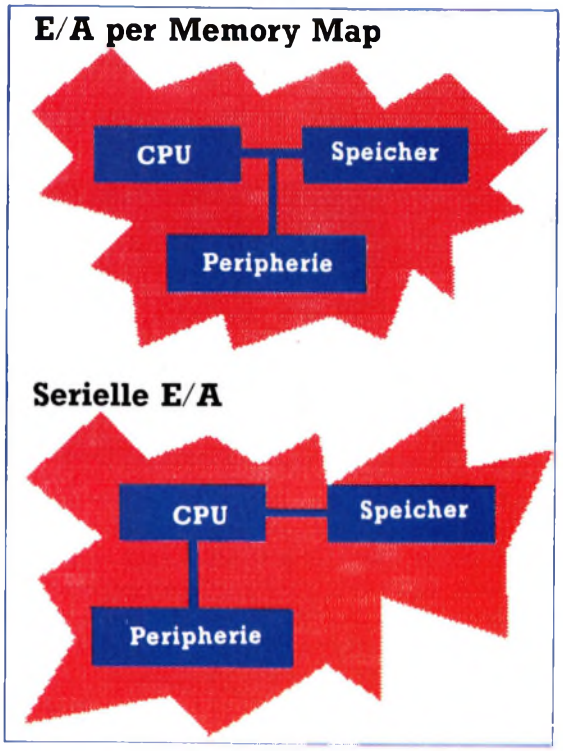
- Per „Memory Map": Hier werden die E/A-Geräte über entsprechende elektronische Schaltungen mit dem zentralen Datenbus (an den auch Speicher und CPU angeschlossen sind) verbunden. Der Prozessor spricht die Geräte dabei wie RAM-Speicher im „Memory Map"-Verfahren an. Wenn mehrere Peripheriegeräte den Bus gleichzeitig beanspruchen, verliert das System allerdings an Geschwindigkeit.
- Isolierte E/A: Diese Methode arbeitet mit einem separaten Bus für E/A-Geräte. Hier ist zwar die Übertragungsgeschwindigkeit höher, doch braucht jedes einzelne Peripheriegerät ein eigenes Befehlsmodul.

Es überrascht nicht, daß der 68000 mit der Memory Map arbeitet. In einer Reihe über die Struktur der Microcomputer hatten wir das Prinzip der E/A-Steuerung von Peripheriegeräten per Memory Map schon ausführlich beschrieben.

Hier die drei wichtigsten Informationen, die der Prozessor für das Memory-Map-System braucht:

- Status: Da Peripherievorgänge Zeit brauchen (beispielsweise zum Drucken eines Zeichens oder für die Tastaturabfrage), verhindern Statusinformationen, daß das Gerät vor Beendigung eines Befehls neue Anweisungen erhält. In diese Kategorie fallen auch Informationen, die nicht unmittelbar mit dem laufenden Vorgang zu tun haben, aber Auskunft über Ablaufart oder Fehler geben.
- Steuerung: Mit Steuerregistern werden Peripheriegeräte eingestellt (konfiguriert) oder aktualisiert (z. B. die Anweisung an ein Diskettenlaufwerk, einen Datenblock zu lesen, oder das Einschalten einer bestimmten Baudrate für einen Übertragungskanal).
- Daten: Schließlich werden Daten an ein Gerät gesandt, von dort empfangen und bis zum Abschluß des Vorgangs zwischengespeichert.

Auf der Ebene der Impulsübermittlung des Hauptdatenbusses können die Vorgänge in den Schnittstellenchips schon recht komplex



Das Bild zeigt die Unterschiede zwischen der seriellen E/A, die über einen eigenen Peripheriebus Daten übermittelt, und der E/A per „Memory Map", bei der die Peripheriegeräte ihre Daten aus dem gleichen Bus erhalten, mit dem auch der Speicher angesprochen werden kann.

werden. Diese komplizierte Steuerung nehmen dem Programmierer normalerweise jedoch spezielle Interfacechips ab. Motorola bietet zwei dieser Chips an – ACIA für die serielle und PIA für die parallele Steuerung.

Serielle Geräte brauchen nur vier Leitungen zum Chip – zwei zum Senden und zwei zum Empfangen von Informationen. Die Daten werden hier nacheinander und bitweise übermittelt, während sie bei der parallelen Kommunikation im Byteformat ankommen.

Die Schnittstellenchips bieten große Flexibilität beim Hardwareaufbau. Dabei kann aber auch die Bitzahl eines Steuerwortes recht groß werden. So werden beispielsweise die ACIA-Steuerbits folgendermaßen zugeordnet:

Bit 0 und 1	Takt-Teilungsfrequenz und Initialisierung (serielle Bitrate)
Bit 2 bis 4	serielles Zeichenformat (z. B. Parity Bit, Zahl der Stopbits)
Bit 5 und 6	Steuerbits für die Übertragung (z. B. Interrupt AN)
Bit 7	Steuerbit für den Empfang auf Interrupt AN gesetzt.

Diese Bits regeln im wesentlichen die Datenübertragung zwischen Computer und Peripherie. Ohne sie ist keine geregelte Kommunikation möglich.

Jede serielle Datenübertragung braucht zwei Befehle. Ein Befehl stellt die ACIA auf die Anforderungen des Peripheriegerätes ein. Hier ein Beispiel:

```
MOVE.B #$3,ACIACON    anfänglicher Hard-
                      warereset
```

```
MOVE.B #$15,ACIACON  Hardware konfigu-
                     rieren
```

In diesem Fall muß ACIACON natürlich zuvor definiert werden.

Auf ähnliche Weise werden auch die Statusbits für die Datenübermittlung gelesen:

Bit 0	Im Empfangsregister sind Daten (ein neues Zeichen steht bereit)
Bit 1	Übertragungsregister ist leer (ein neues Zeichen kann gesendet werden)
Bit 2 und 3	Signale zur Modemsteuerung (CTS und DCD)
Bit 4 bis 6	Fehleranzeige für Empfangsdaten (z. B. bei einem Parityfehler)
Bit 7	Interruptanforderung

Da Statusregister nur gelesen werden und in Steuerregistern nur gespeichert wird, brauchen wir in der Memory Map auch nur eine Adresse. Dies trifft auch auf Datenregister zu, da daraus nur die empfangenen Daten gelesen oder Sendedaten geschrieben werden.

Hier ein Beispiel für die Eingabe:

```
INCH   BTST     #0,ACIASTAT   steht ei
                              chen zu
                              fügung:
       BEQ      INCH          Falls nic
                              nochma
                              prüfen
       MOVE.B   ACIADATA,DO   Zeicher
                              Parame
                              DO lese
       RTS                    Rücksp
                              von der
                              routine
```

Das Programm wartet, bis Bit 0 des Steuerregisters auf 0 gesetzt ist und liest dann das Datenregister. ACIASTAT entspricht ACIACON.

Auf die gleiche Weise kann eine Subroutine auch Zeichen ausgeben:

```
OUTCH  BTST     #1,ACIASTAT   warten,
                              das Dat
                              gister le
       BEQ      OUTCH         Falls nic
                              nochma
                              verzwei
                              und prü
       MOVE.B   DO,ACIADATA   Zeichen
                              geben
       RTS
```

Zur Wiederholung muß danach nur folgende Subroutine aufgerufen werden:

```
ECHO   JSR      INCH          Zeichen
                              lesen
       JSR      OUTCH         Zeichen
                              geben
       BRA      ECHO          und Abl
                              wiederh
```

Da der Computer die Daten zeichenweise übermittelt, ist dieser E/A-Ablauf nicht besonders schnell, zeigt aber den Grundmechanismus der E/A-Programmierung.

Die Befehle des 68000

Bevor wir uns mit den Interrupts des 68000 beschäftigen, geben wir einen Überblick über alle bisher beschriebenen Befehle. Hier zunächst die Gruppe der Kopierbefehle, bei denen die MOVE-Anweisung eine wichtige Rolle spielt:

Befehl	Vorgang
MOVE	Transportiere von der Quelle zum Ziel
MOVEM	Transportiere mehrere Daten und Adreßregister (praktisch für die Parameterübergabe bei Subroutinen)
MOVEA	Transportiere den Inhalt einer Adresse in ein Adreßregister
LEA	Lade die Quelladresse in ein Adreßregister
MOVEQ	Schnellübertragung einer kleinen Konstanten in ein Datenregister
PEA	Schiebe effektive Adresse auf den Stapel
SWAP	Vertausche untere und obere Hälfte eines Datenregisters
EXG	Tausche die Daten zweier Register
LINK/UNLK	Wird zur Stacksteuerung für die Parameterübergabe bestimmter Subroutinen eingesetzt

Dieser Befehlssatz bietet zwar viele flexible Möglichkeiten, doch sollten Sie besonders bei weniger gebräuchlichen Anweisungen wie LINK/UNLK vorsichtig sein. Die Befehle für die Ganzzahlenarithmetik:

Befehl	Vorgang
ADD	Quelle auf Ziel addieren
ADDA	Quelle auf das Zieladreßregister addieren
ADDI	Unmittelbare Daten auf das Ziel (datenveränderndes Register) addieren
ADDQ	Schnelladdition für kleine Konstante (1 bis 8)
ADDX	Addieren mit Übertrag
SUB (A/I/Q/X)	Subtrahiere Quelle vom Ziel in einer der Adreßvarianten, wie bei ADD gezeigt
CMP(A/I)	Vergleiche Quelle mit Ziel und setze die Bedingungscodes
CMPM	Vergleiche Speicherstellen unter Nach-Inkrementierung des Pointers
NEG(X)	Negiere das Operanden-(Daten-) Register
EXT	Vorzeichenerweiterung des Operanden-(Daten-)Registers
MULU	Multipliziere Daten ohne Vorzeichen in der angegebenen effektiven Adresse, und speichere das Ergebnis im Datenregister
MULS	Wie bei MULU, jedoch für Daten mit Vorzeichen
DIVU(S)	Dividiere Ziel-(Daten-)Register

	durch Quelloperanden. Quotient und Rest werden im Zielregister gespeichert
TST	Setze die Bedingungscodes je nach Operand
TAS	Teste und setze das höchstwertige Bit des Operanden
CLR	Setze Operand auf 0

Bei diesen breitgefächerten Anweisungen müssen Sie darauf achten, daß Sie für den Operanden den richtigen Adreßmodus einsetzen. Es folgen die Befehle der BCD-Arithmetik:

Befehl	Vorgang
ABCD	Addiere BCD-Operanden und speichere die Summe im Ziel
SBCD	Subtrahiere die Operanden
NBCD	Negiere den BCD-Operanden

Beachten Sie, daß es keinen BCD-Befehl für Multiplikation gibt. Die folgenden Logikbefehle sind jedoch umfassend:

Befehl	Vorgang
AND	Logisches AND der Quelle mit dem Ziel. Mindestens ein Operand muß Datenregister sein
ANDI	Logisches AND mit unmittelbaren Daten als Quelle
OR	Logisches OR mit den gleichen Komponenten wie AND
ORI	Logisches OR mit unmittelbaren Daten
EOR	Exklusives OR
EORI	Exklusives OR mit unmittelbaren Daten
NOT	Einerkomplement bilden

Folgende Befehle steuern das bitweise Testen und Bearbeiten:

Bitverarbeitende Befehle

Befehl	Vorgang
BTST	Das im Quelloperanden festgelegte Bit testen
BSET	Zieloperand testen und Bit setzen
BCLR	Bit testen und löschen
BCHG	Bit testen und Operand ändern

Beachten Sie, daß sie mit dem getesteten Bit keine Abläufe ausführen müssen.

Befehle für Verschiebung und Rotation

Befehl	Vorgang
ASI	Arithmetische Linksverschiebung
ASR	Arithmetische Rechtsverschiebung
LSL	Logische Linksverschiebung
LSR	Logische Rechtsverschiebung
ROL	Operand nach links rotieren und Übertragsbit entsprechend setzen
ROR	Operand nach rechts verschieben und Übertragsbit entsprechend setzen

Schließlich noch die Befehle zur Steuerung von Programmen und Subroutinen:

Befehle zur Programmsteuerung

Befehl	Vorgang
BRA	Unbedingte Verzweigung
JMP	Unbedingter Sprung (sehr praktisch, wenn Sie die Sprungadresse zuvor berechnen wollen)
Bcc	Bedingte Verzweigung. Abhängig vom Testen des mit "cc" angegebenen Bedingungscodes. Hier die Codes im einzelnen:

Für Operanden mit Vorzeichen:

GT	größer als
LT	kleiner als
GE	größer oder gleich
LE	kleiner oder gleich
VS	Überlaufbit gesetzt
VC	Überlaufbit nicht gesetzt

Für Operanden ohne Vorzeichen:

EQ	gleich
NE	ungleich
MI	Minus
PI	Plus
HI	größer als
LS	kleiner oder gleich
CS	Übertragsbit gesetzt
CC	Übertragsbit nicht gesetzt
DBcc	Zähler dekrementieren und verzweiger Bedingung 'cc' (wie oben)

Befehle zur Subroutinensteuerung

Befehl	Vorgang
JSR	Auf Subroutine springen
BSR	Verzweige auf Subroutine
RTS	Rücksprung von der Subroutine

Diese Zusammenfassung enthält nicht alle Befehle des 68000. So gibt es für Betriebssystemdesigner weitere Anweisungen zur Systemsteuerung, die außerhalb der Thematik dieses Kurses liegen. Diese Befehle drehen sich hauptsächlich um das Statusregister, die Stackpointer und die ('Trap' genannten) 'Softwareinterrupts'. Sie sind im Anwenderhandbuch des 68000 ausführlich beschrieben.

System-unterbrechung

In der letzten Folge über den Motorola 68000 hatten wir uns die serielle Ein- und Ausgabe angesehen. Nun wenden wir uns den parallelen Schnittstellen zu und stellen Ihnen hier die Interruptbehandlung vor.

Sehen wir uns zunächst an, wie die serielle Ausgabe in der Subroutine „Meldung" aussieht. Die Adresse des Textes wird der Subroutine mit dem Adreßregister A3 übergeben:

```
LEA  TEXT,A3     Pointer auf den Text der Meldung setzen
JSR  MELDUNG     Meldung ausgeben
```

Das Modul gibt einen Text aus, der als

```
TEXT:DC.B        'Computer Kurs',$00
```

gespeichert wurde. DC.B ist eine Assembleranweisung, die Platz für den Text „Computer Kurs" reserviert. Das „leere" Byte $00 ist eine Endmarkierung, die anzeigt, daß kein weiterer Text folgt.

Die Routine zur byteweisen Ausgabe des Arrays TEXT arbeitet mit der Subroutine OUTCH. Dabei zeigt das ‚Ready'-Bit an, daß ACIA sendebereit ist und ein weiteres Zeichen ausgeben kann. Die Subroutine „Meldung" könnte so aussehen:

```
MELDUNG: MOVE.B (A3)+,DO   nächstes Byte der Meldung holen
         BEQ    FERTIG     auf Ende der Meldung testen
         JSR    OUTCH      Zeichen ausgeben
         BRA    MELDUNG    Schleife bis zum Textende wiederholen
FERTIG:  RTS
```

Die Bytes der Meldung werden in DO kopiert (das Datenregister, das OUTCH als Wertregister verwendet), wobei A3 ein Pointer mit nachinkrementierter indirekter Adressierung ist. Wenn das Byte auf Null steht, wird über das Label FERTIG der Rücksprung ausgelöst, anderenfalls gibt OUTCH das Zeichen aus.

Die parallele Datenausgabe (das gleichzeitige Senden von Datenbits) geschieht mit dem PIA-Chip. Da dieser Chip auf ein breites Anwendungsspektrum ausgelegt ist, muß er erst einmal auf die individuelle Hardwarekonfiguration eingestellt werden. Dies entspricht auf dem ACIA-Chip dem Setzen von Bitraten und Byteformaten.

Beim PIA-Chip müssen die acht parallelen Leitungen beider Chiphälften (A und B) als Ein- oder Ausgabeleitungen (d. h. die Datenrichtung) konfiguriert werden. Das Schreiben auf Bit 2 des Steuerregisters (CRA oder CRB) zeigt an, daß der Inhalt eines Datenrichtungsregisters (DDRA oder DDRB) gesetzt werden soll. Hier ein Beispiel:

```
CONFIGPIA: CLR.B   PIACRB      Bit des Steuerregisters auf Null setzen
           MOVE.B  #$FF,PIADDRB Datenrichtung auf Ausgabe setzen
           BSET    #2,PIACRB  Auf normales Datenregister zurückgehen
           CLR.B   PIACRA     Datenrichtungsregister für A adressieren
           CLR.B   PIADDRA    Datenrichtung auf Eingabe setzen
           BSET    #2,PIACRA  Auf normales Datenregister zurückgehen
```

Dabei wird die gesamte Seite A auf Eingabe und Seite B auf Ausgabe gesetzt. Die eigentliche Datenübertragung zu parallel angeschlossenen Geräten geschieht dann mit:

```
MOVE.B DO,PIADRA Inhalt von DO ausgeben
```

während folgende Zeile die Eingabe erledigt:

```
MOVE.B PIADRA,D1 Eingabe in D1 einlesen
```

Alle angesprochenen PIA-Adressen müssen zuvor wie folgt initialisiert werden:

```
PIADRA EQU $30051
PIACRA EQU $30053
```

Die Interpretation der von der PIA gelesenen oder geschriebenen Daten hängt von dem Ge-

rät ab, das mit den digitalen Leitungen des Chips verbunden ist. Es könnte zum Beispiel ein siebenteiliges Anzeigeelement für Dezimalzahlen angeschlossen sein (siehe Bild). Die Anzeige der Zahl 3 beispielsweise müßte die Elemente 1, 4, 2, 5 und das Element 3 mit Strom versorgen.

Es müssen aber nicht nur Daten zum Peripheriegerät übertragen, sondern auch dessen elektrische Funktionen gesteuert werden. Beim Einspeisen der Daten in die siebenteilige Anzeige übernehmen einige der freien Bits daher Steuerfunktionen. Da die Steuersignale oft einen elektrischen Taktpuls simulieren, muß auch das Steuerbit gesetzt und wieder auf Null gestellt werden:

```
BSET  #CONBITNO,PIADRA  Steuerbit setzen
JSR   DELAY             kurze Warteroutine
BCLR  #CONBITNO,PIADRA  Bit danach wieder auf Null stellen
```

Dieses kurze Modul erzeugt das Taktsignal auf dem digitalen Kanal, der CONBITNO (bei Adresse PIADRA) zugeordnet ist.

Interrupts sollen im allgemeinen helfen, die CPU optimal auszunutzen und eine Reaktion auf externe Ereignisse zu ermöglichen. So braucht die CPU während der Zeichenausgabe (z. B. beim Drucken mit OUTCH) nicht zu pausieren. Wir müssen dabei jedoch wissen, wann der Druck beendet ist und das nächste Zeichen ausgegeben werden soll.

Beim Warten auf Eingaben ist die Situation nicht ganz so einfach. Hier hängt die Schnelligkeit des Systems zwar viel von der Eingabegeschwindigkeit ab, aber auch von anderen Aufgaben, die die CPU ausführt, z. B. die parallele Druckausgabe während des Wartens.

Für diesen parallelen Vorgang muß die logische Reihenfolge der Ereignisse so organisiert werden, daß weder die Programmsteuerung noch Daten verlorengehen. Sehen wir uns die Abläufe einmal genauer an.

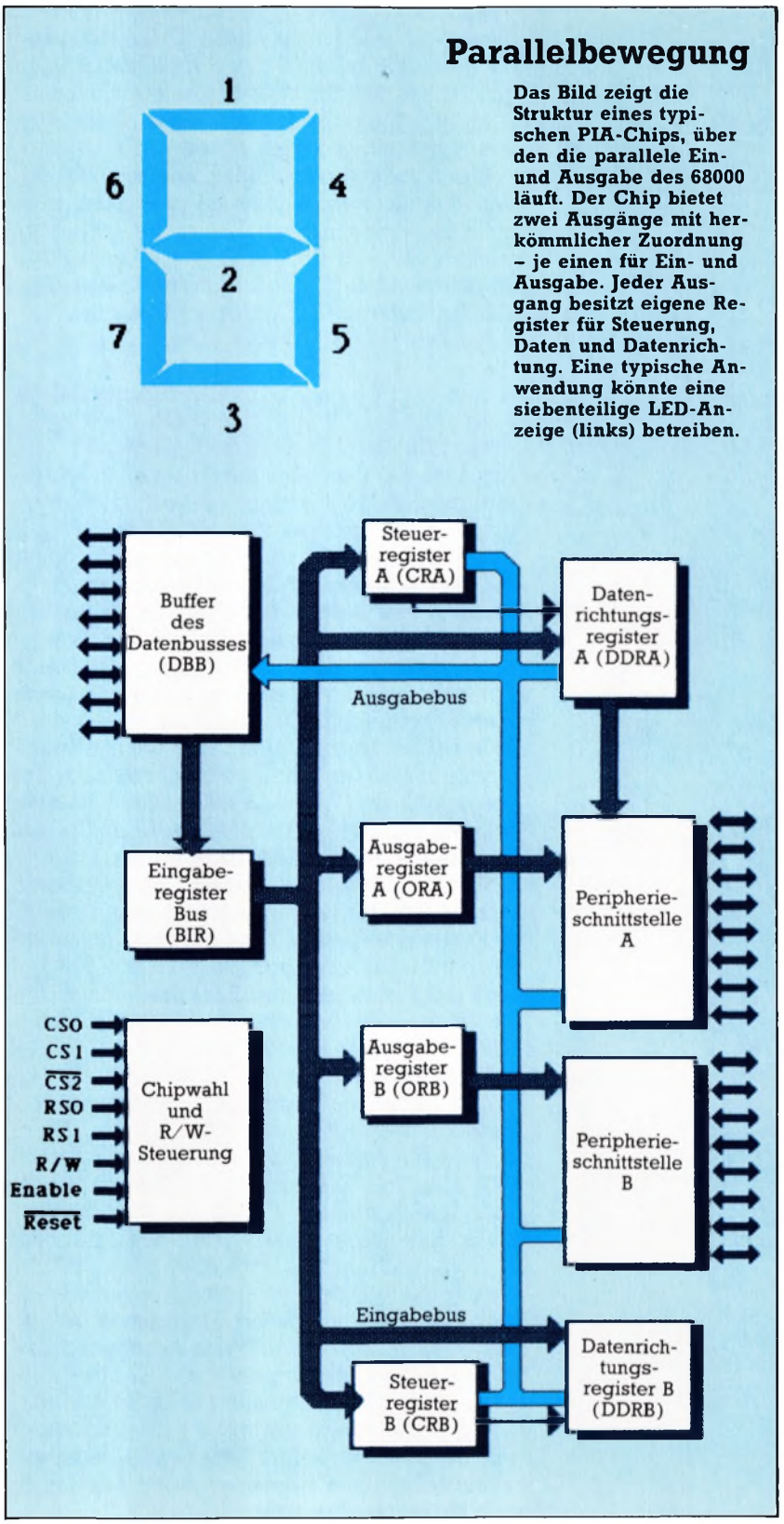
- Computerstatus sichern. Damit wird sichergestellt, daß Sie jederzeit problemlos und ohne Datenverluste zum Programm zurückkehren können. Zunächst müssen wir jedoch definieren, woraus der „Status" eines Computers überhaupt besteht.

Als Computerstatus ließe sich der gesamte Programm- und Datenbereich eines Anwenderprogramms mit Registern und Programmzähler definieren. Da Interrupts in der Praxis jedoch kaum die Anwenderprogramme ändern, genügt das Sichern des Programmzählers (PC) und des Statusregisters (SR). Diese beiden Register liefern genaue Informationen über den Status des unterbrochenen Programms.

- Interruptquelle identifizieren. Da fast alle Computer mit mehr als nur einem Peripheriegerät arbeiten, müssen wir feststellen, welche Bearbeitungsroutine gemeint ist.
- Die Interrupts anderer Quellen abschalten. Das Abschalten ist notwendig, da Interruptanforderungen, die während der Bearbeitung eines Interrupts entgegengenommen werden, zu Datenverlusten führen können. Diese Situation tritt jedoch nur ein, wenn die Bearbeitung des zweiten Interrupts länger dauert als die

Parallelbewegung

Das Bild zeigt die Struktur eines typischen PIA-Chips, über den die parallele Ein- und Ausgabe des 68000 läuft. Der Chip bietet zwei Ausgänge mit herkömmlicher Zuordnung – je einen für Ein- und Ausgabe. Jeder Ausgang besitzt eigene Register für Steuerung, Daten und Datenrichtung. Eine typische Anwendung könnte eine siebenteilige LED-Anzeige (links) betreiben.



Zeit, die zwischen zwei Datenübertragungen des ersten Interrupts liegt.

- Die Bearbeitungsroutine des Interrupts aktivieren. Diese Aufgabe kann automatisch oder über einen Spezialbefehl erfolgen.
- Rücksprung auf das unterbrochene Programm. Nach der Interruptbearbeitung wird der ursprüngliche Zustand wiederhergestellt.

Wie sieht dieser logische Ablauf nun auf dem 68000 aus? Zuerst wird der Maschinenstatus gesichert, indem PC und SR auf den Stack geschoben werden. Da dieser Mechanismus Interrupts schachteln kann, lassen sich auch Interruptroutinen unterbrechen.

Wenn die Interruptroutine Register verwendet, die für andere Zwecke reserviert sind, kann sie diese Register beim Einsprung sichern und beim Rücksprung wieder in ihren ursprünglichen Zustand versetzen. Auf dem 68000 erledigt MOVEM diesen Vorgang:

```
MOVEM D1,D3,—(SP)   Datenregister auf den
                    Stack schieben
          |         Bearbeitungscodes für
                    Interrupts
MOVEM(SP)+,D1,D3    wiederherstellen
```

Auf dem 68000 ist auch das Feststellen der Interruptquelle sehr einfach, da jedem Interrupt eine eindeutige Speicherstelle ('Vektor' genannt) zugewiesen ist. Vektoren sind Pointer auf die Adressen der Bearbeitungsroutinen. Da mehrere Geräte den gleichen Vektor ansprechen können, müssen die entsprechenden Geräte abgefragt werden, um die Quelle der Interruptanforderung herauszufinden. Dies kommt auf dem 68000 jedoch nur selten vor.

Auch die Interruptprioritäten der einzelnen Geräte regelt der 68000 automatisch nach der vorgegebenen Hardwareeinstellung. Dabei dringen nur Interrupt von höherer Priorität (als der gerade bearbeitete) zur CPU vor.

Wenn ein Interrupt von der CPU bearbeitet werden soll, wird der PC mit dem Inhalt des Interruptvektors geladen und die entsprechende Bearbeitungsroutine aufgerufen. Die Routine lädt dann entweder die Eingabedaten in ein Register – und von dort in einen Zwischenspeicher – oder sendet Daten aus einem Buffer zum Ausgabegerät.

Der Rücksprung zum unterbrochenen Programm sieht zwar wie ein normaler Rücksprung aus einem Submodul aus, wird statt mit RTS aber mit RTE ausgeführt. Da RTE dabei automatisch den PC und SR aus dem Systemstack lädt, sollten alle Interruptroutinen mit RTE enden. Das bedeutet aber auch, daß A7 (der Pointer für den Systemstack) auf die Register des unterbrochenen Programms zeigen muß. Wenn Sie Daten auf dem Systemstack abgelegt haben, müssen Sie diese vor der Ausführung der RTE-Anweisung unbedingt wieder herunterziehen. Besser wäre es, einen anderen Stack zu verwenden. Der 68000 kann zur Veränderung von Adressen jedes Adreßregister als Stack einsetzen.

Sehen wir uns nun eine typische Interruptroutine an, die die Geräte einzeln abfragt („Polling"), um die Interruptquelle festzustellen. Unser Beispielsystem enthält auf „Ebene 4" (eine Hardwareleitung mit Priorität 4) zwei mögliche Interruptquellen. Die Geräte sind 1) eine externe Tastatur und 2) eine Echtzeituhr. Die Uhr zählt Sekunden und wird von anderen Teilen des (Software-)Systems als Taktgeber eingesetzt.

Um den Interruptvektor für Ebene 4 (bei $70) richtig setzen zu können, müssen wir ihn zuvor initialisieren:

```
MOVE.L  #EXTDEVS,$70    Vektor der Ebene
                        4 setzen
```

Natürlich werden auch andere Register gesetzt, beispielsweise die Stacks für Anwender und System:

```
MOVE.L #STACK,SP        Systemstack set-
                        zen
MOVE.L #USERSTACK,A0    Anwenderstack
                        setzen
```

A7 steht nun auf der Adresse STACK und A0 auf USERSTACK.

Hier die Interruptbearbeitungsroutine:

```
EXTDEVS MOVEM.L  D0—D7,(A0)   Datenre-
                              gister si-
                              chern
        BTST     #7,PIACRA    feststel-
                              len, ob
                              Tastatur-
                              interrupt
                              vorliegt
        BNE      CHAR
CKCLK   BTST     #6,PIACRB    falls
                              nicht,
                              muß es
                              die Uhr
                              sein
        BNE      CLOCK
        BRA      WILD         anson-
                              sten liegt
                              ein Feh-
                              ler vor
CHAR    JSR      KEYBOARD     Tastatur-
                              routine
                              aufrufen
        BRA      CKCLK        die Uhr
                              könnte
                              ebenfalls
                              unterbro-
                              chen ha-
                              ben
CLOCK   JSR      SECS         Uhrrou-
                              tine auf-
                              rufen
WILD
        MOVEM.L  (SP)+,D0—D7  Datenre-
                              gister
                              wieder-
                              herstel-
                              len
        RTE
```

Diese Routine sichert alle Datenregister.

Es kommen neue Schneider-Befehle

Zum Abschluß unserer Serie über das Schneider-Betriebssystem sehen wir uns die Systemerweiterungen an, mit denen neue Befehle an das Locomotive BASIC angefügt werden können.

Die Kernroutinen sind das Herz des Schneider-Betriebssystems. Sie erledigen die interne Verwaltung des Computers und steuern Ereignisse, Interrupts und den Speicher.

Die Kernroutinen werden zumeist mit Adressen zwischen &BCC8 und &BD10 angesprochen. Sie haben jedoch auch einen eigenen Jumpblock, der in einen oberen (&B9000 – &B921) und unteren Bereich (&0000 – &003B) geteilt ist. Im Gegensatz zum Haupt-Jumpblock des Firmwarebereiches läßt er sich jedoch nicht vom Anwender patchen. Einige dieser Routinen sind außerordentlich praktisch. Wir haben sie in einer Tabelle zusammengestellt.

Die Kernroutinen können unter anderem externe Befehle bearbeiten, die sich im RAM oder in den ROMs befinden. Befehle, die ins RAM geladen werden, heißen „Resident System Extension" (RSX). Externe Befehle können beliebig lang sein und jede Funktion übernehmen. So sind die AMSDOS-Befehle |DIR und |ERA beispielsweise Routinen zur Bearbeitung von Diskettendateien. Auch das gesamte BASIC arbeitet als RSX – geben Sie |BASIC ein, und beobachten Sie, was da passiert. Externe Befehle lassen sich auf zwei Arten anlegen. Erstens als Befehle, die beim Einschalten aus dem ROM geladen werden, und zweitens als programmgesteuerte Befehle, die aus RAM oder ROM geladen werden. In beiden Fällen werden die Befehle über eine Befehlstabelle angesprochen, deren Format im nebenstehenden Kasten beschrieben ist. Die Befehlsnamen können bis zu 16 Zeichen lang sein, wobei Bit 7 des letzten Zeichens gesetzt sein muß. Die Zeichen, die Sie in Ihren Computer eingeben, dürfen jeden Sieben-Bit-Code annehmen.

Anmeldung

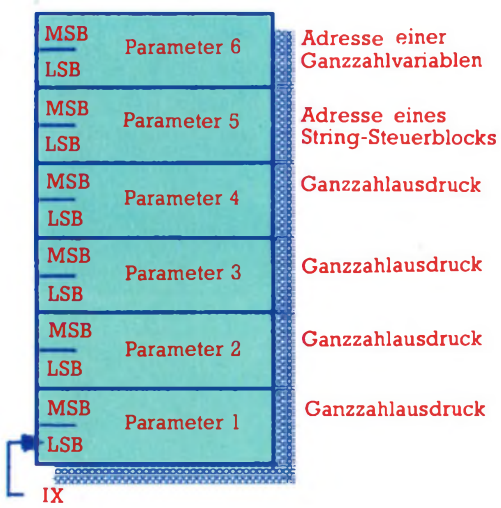
RSX-Befehle (d. h. ins RAM geladene Befehle) werden mit der Kernroutine KL_LOG_EXT „angemeldet" (d. h. ihre Existenz der Firmware angezeigt). Kl_LOG_EXT wird über &BCD1 mit der Adresse der Befehlstabelle in BC aufgerufen. HL muß die Adresse eines Vier-Byte-Arbeitsbereiches enthalten, den die Kernroutine braucht. Beide Adressen dürfen nur in den mittleren 32K des RAM liegen.

Bei Befehlen, die im ROM untergebracht sind, wird der Firmware das ROM und nicht die Befehlstabelle angemeldet. Das ROM muß dabei dem im Bild gezeigten Format entsprechen. Es können Vordergrund-, Hintergrund- oder Erweiterungs-ROMs definiert werden. Vordergrund-ROMs enthalten Programme, die beim Aufruf die Steuerung der Firmware übernehmen. Beispiele sind das BASIC-Rom oder die ROMs anderer Sprachen wie PASCAL.

Hintergrund-ROMs enthalten externe Befehle, die vom Vordergrundprogramm nicht angemeldet zu werden brauchen. In Erweiterungs-ROMs wird der Code von Vordergrundprogrammen oder externen Befehlen untergebracht, der nicht mehr in das entsprechende ROM paßt.

Jedes ROM wird im Bereich von 0 bis 251 adressiert, wobei bestimmte Typen Einschränkungen unterworfen sind. So können Vordergrund-ROMs alle Adressen annehmen, vorausgesetzt, die darunterliegenden Adressen sind von anderen ROMs belegt (die Kernroutine sucht die ROMs von Adresse 0 an aufwärts, bis sie den ersten freien Platz findet). Hintergrund-ROMs werden auf dem Commodore# PC 464 in den Adressen 1–7 eingebaut und in 1–15 auf den beiden Schneider-Computern 664 und 6128.

Die Adressierung der Parametertabelle



Beim Einsprung in eine Befehlsroutine, die vom BASIC aus aufgerufen wird, werden die Parameter in einem Parameterblock gespeichert. Das Register IX zeigt dabei auf die Basisadresse des Blocks, während das Register A die Anzahl der Parameter übergibt. Das Bild zeigt die Struktur eines Blocks, der nach der Ausführung von |COMMAND,5,24.6,a%, x,@bS, @x% angelegt wurde.

Wenn beispielsweise FORTH als ROM 0 eingebaut ist und sich bei ROM 1 ein Hintergrund-ROM befindet, sollte ein zweites Vordergrund-ROM bei der ROM-Adresse 2 installiert werden, um die Kontinuität sicherzustellen. Ein zweites Hintergrund-ROM könnte allerdings in die Adressen zwischen 3 und 7 (15 auf dem 664/6128) eingesetzt werden und ein Erweiterungs-ROM in jede beliebige Adresse.

Das Anschalten der Maschine meldet automatisch alle Vordergrund-ROMs an. Hintergrund-ROMs können dann entweder mit KL_INIT_BACK einzeln oder mit KL_ROM_WALK insgesamt initialisiert werden. Das Ansprechen des BASIC-ROMs initialisiert automatisch alle Hintergrund-ROMs.

Da die Erweiterungs-ROMs den gleichen Speicherbereich belegen wie das eingebaute BASIC-ROM (&C000 bis &FFFF), muß vor deren Einsatz das BASIC-ROM abgeschaltet und das gewünschte ROM angeschaltet werden. Es gibt mehrere Kernroutinen zum An- und Abschalten der oberen und unteren ROMs, mit denen ein Programmierer die Hardware nicht direkt ansprechen muß.

Nach der Anmeldung lassen sich die externen Befehle entweder direkt vom BASIC aus oder über die Kernroutine KI_FIND_COMMAND aufrufen. Dieser Aufruf durchsucht die RSX- und die Hintergrund-ROMs nach einem Befehlsnamen. Wird er gefunden, erscheint die Einsprungadresse in HL, und das Register C enthält die Nummer des Hintergrund-ROMs (wenn die Befehlsroutine im RAM liegt, kann C ignoriert werden). Wird kein Befehl gefunden, liefert die Routine im Übertrag ein FALSCH. KI_FIND_COMMAND wird über &BCD4 aufgerufen, HL enthält dabei die Adresse des Befehlsnamens, bei dessen letztem Zeichen Bit 7 gesetzt sein muß.

Speicherplatz

Externe Befehle lassen sich leicht vom BASIC aus aktivieren, indem vor den Befehlsnamen das Symbol | (Shift @) gestellt wird. BASIC kann dabei auch Parameter übergeben, die allerdings durch Kommas getrennt sein müssen. Hier ein Beispiel:

```
|COMMAND,1,2,3,x,y*3,@$,@X%
```

Die Firmware übergibt die Steuerung an die Befehlsroutine, wobei Register A die Anzahl der Parameter enthält und IX auf eine Tabelle mit Parameterinformationen zeigt. In dieser vorliegenden Tabelle sind jedem Parameter jeweils zwei Bytes zugeordnet, die folgendes Format besitzen:

Ganzzahl oder Ganzzahlausdruck — 16-Bit-Ganzzahlen mit Vorzeichen
Reelle Zahlen — 16-Bit-Ganzzahlen ohne Vorzeichen

Auch die Adressen von String- oder numerischen Variablen können mit dem Symbol @ übergeben werden. Ganzzahlvariablen hinterlassen dabei die Zwei-Byte-Adresse ihres Speicherplatzes und Stringvariablen die Adresse des String-Steuerblocks. Leider läßt sich nicht feststellen, ob ein Tabelleneintrag die Adresse eines String-Steuerblocks enthält oder eine absolute Ganzzahl. Hier muß die Befehlsroutine entscheiden, wie der Parameter interpretiert werden soll.

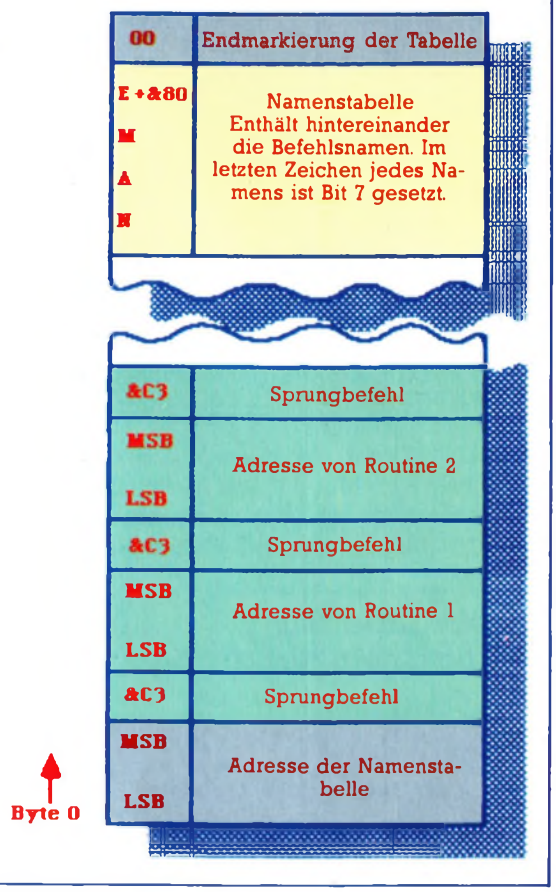
Beim Einsprung in die Befehlsroutine zeigt IX auf das LSB des letzten angegebenen Parameters (im obigen Beispiel ist dies die

Die Adressen der Kernroutinen

&BCC8 RESET	Reset für Kernroutinen, löscht Ereignisse, Warteschlangen etc.
&BCCB ROM WALK	Initialisiert alle Hintergrund-ROMs
&BCD1 LOG EXT	RSX-Befehlstabelle anmelden
&BCD4 FIND COMMAND	RSX suchen; Befehlsadresse im Hintergrund- oder Vordergrund-ROM finden
&B90F ROM SELECT	Ein bestimmtes oberes ROM anwählen
&B912 CURR SELECTION	Testen, welches obere ROM gerade angewählt ist

Externe Befehle werden über eine Befehlstabelle aufgerufen. Eine Tabelle enthält die Daten einer Reihe von unterschiedlichen Befehlen. Das Bild zeigt die Struktur einer derartigen Tabelle.

Tabelle für externe Befehle



Adresse von X%). Die Inkrementierung von IX indiziert dabei das LSB des vorletzten Parameters. Unsere Graphik zeigt, wie die Tabelle bei unserem Beispiel indiziert wird.

Oft empfiehlt es sich, zuerst den ersten Parameter anzusehen und dann die Parameter in Eingabereihenfolge abzurufen (nicht umgekehrt). Die Routine unseres ersten Listings ändert IX, das dann auf das LSB des ersten Parameters zeigt. Für den zweiten Parameter wird dann IX einmal dekrementiert. Für das LSB des zweiten Parameters muß IX dann nochmals dekrementiert werden.

RSX-Befehle sind für den Maschinencodeprogrammierer recht einfach einzusetzen, da sie nicht – wie bei vielen anderen Maschinen – über CALL- oder SYS-Befehle aufgerufen werden müssen. Aus diesem Grund hat der Befehl CALL auf den Schneider-Computern nur untergeordnete Bedeutung, es zahlt sich die Einrichtung von Systemerweiterungen aus.

Aufbau eines Erweiterungs-ROMs

Die Tabelle zeigt die Bytestruktur, mit der das Betriebssystem ein einzelnes Erweiterungs-ROM identifiziert und „anmeldet". Eine spätere Anfrage an das Betriebssystem liefert die Adressen der Befehlsroutinen, die in der Befehlstabelle des externen ROMs definiert sind.

Übergabe der Programmsteuerung

Parameterzeiger

Diese kurze Routine ändert IX beim Einsprung in eine externe Befehlsroutine, so daß sie auf den ersten Eingabeparameter zeigt:

```
        ;entry: A contains number of parameters
        ;       IX contains address of table
        ;exit:   IX   contains              the address
        ;        of   1st                   entry in table
        ;       DE corrupt, other registers
        ;       preserved
        ;
5F              ld   e,a                ;copy number to E
1D              dec  e                  ;no of parameters-1
CB23            sla  e                  ;*2
1600            ld   d,0                ;get offset in DE
DD19            add  ix,de              ;add it to start
        ;IX now points to LSB of 1st entry
        ;in the parameter table
```

Firmwareaufruf

Dieses Listing enthält einen RSX, über den die Firmware (mit entsprechend gesetzten Registern) vom BASIC aus direkt aufgerufen werden kann. Die Befehlssyntax sieht folgendermaßen aus:

|FIRMCALL, <entry address>, [,<A> <HL> [,<BC>[, DE]]]]

```
        kl_log: equ  #bcd1
01902C  logon:  ld   bc,commands        ;point to table
219E2C          ld   hl,scratch         ;and scratchpad
CDD1BC          call kl_log             ;log on command
C9              ret                     ;and that's it
952C    comman: defw name               ;point to name
C3A22C          jp   firmcall           ;entry point
46495240 name:  defm "FIRMCAL"          ;set bit 7 of...
CC              defb "L"+#80            ;..last byte of name
00              defb 0
        scratc: defs 4                  ;reserve data area
        ; IFIRMCALL, entry, a, hl, bc, de
5F      firmca: ld   e,a                ;copy number to E
1D              dec  e                  ;no of parameters-1
CB23            sla  e                  ;*2
1600            ld   d,0                ;get offset in DE
DD19            add  ix,de              ;add it to start
DDE5            push ix
E1              pop  hl
FE06            cp   6                  ; up to 5 parameters
D0              ret  nc                 ;too many, abort
B7              or   a                  ;any parameters?
C8              ret  z                  ; no, abort
47              ld   b,a                ;save count in B
23              inc  hl                 ;get MSB of address
7E              ld   a,(hl)             ;read it in
32E62C          ld   (entry+1),a        ;save it
2B              dec  hl                 ;LSB
7E              ld   a,(hl)
32E52C          ld   (entry),a
2B              dec  hl                 ;MSB of AF
2B              dec  hl                 ;LSB of AF
05              dec  b                  ;got enough?
2822            jr   z,call             ;yes, do the call
7E              ld   a,(hl)             ;else read in A
05              dec  b                  ;enough?
281E            jr   z,call             ;yes, do the call
2B              dec  hl                 ;MSB of HL
56              ld   d,(hl)
2B              dec  hl
5E              ld   e,(hl)
EB              ex   de,hl              ;get into HL
05              dec  b                  ;enough?
2816            jr   z,call             ;yes do the call
EB              ex   de,hl              ;get pointer back
2B              dec  hl                 ;MSB of BC
F5              push af                 ;save A for now
7E              ld   a,(hl)
2B              dec  hl                 ;LSB of BC
4E              ld   c,(hl)
05              dec  b                  ;done enough
47              ld   b,a                ;set up BC anyway
EB              ex   de,hl              ;and HL
2003            jr   nz,miss            ;get the last one
F1              pop  af                 ;restore the stack
1808            jr   call               ;and do the call
F1      miss:   pop  af
EB              ex   de,hl              ;get table back
D5              push de                 ;save hl value
2B              dec  hl                 ;MSB of DE
56              ld   d,(hl)
2B              dec  hl                 ;LSB of DE
5E              ld   e,(hl)
E1              pop  hl                 ;get HL back
C3      call:   defb #c3                ;jump instruction
0000    entry:  defw 0                  ;jump adress
        ;the jump address is patched
        ;into the routine
```

Korrekt übersetzt

In unserer letzten Folge über die Programmierung des 68000 sehen wir uns die Assemblervorgänge genauer an. Wir untersuchen, welche Rolle der Assembler in der Programmierung spielt.

In den letzten Folgen hatten wir schon mehrfach Assembleranweisungen erwähnt, um die Fähigkeiten des 68000 zu verdeutlichen. Dabei gingen wir davon aus, daß der Computer die Befehle so ausführt, wie sie im Assembler erscheinen. Natürlich kann die Maschine nur den binären Code verstehen, den die Assemblerbefehle auf höherer Ebene darstellen, wobei der Assembler diese Anweisungen in den Maschinencode übersetzt.

Für das Verständnis von Programmen ist es praktisch, sich die Ausführung auf der Ebene des Quellentextes vorzustellen. Diese Ebene kann weit oben liegen – zum Beispiel bei einem Anwendungspaket wie WordStar –, etwas niedriger – bei PASCAL-Programmen – oder noch maschinennaher auf der Ebene der Assemblerprogramme. Somit entsteht der Eindruck von zahlreichen „virtuellen" Maschinen auf unterschiedlichen Ebenen.

Programmausführungen

Jede Programmausführung – egal auf welcher Ebene – geschieht jedoch durch

- Übersetzung: Ein Programm auf einer Ebene wird in eine niedrigere Ebene übersetzt. So können Sie ein PASCAL-Programm auf Ebene 3 in den Assemblercode übersetzen, der auf Ebene 2 mit einem einfachen Compiler abläuft.
- Interpretation: Die Programme einer Ebene werden von einem weiteren Programm (‚Interpreter' genannt), das auf einer niedrigeren Ebene läuft, ‚gedeutet'. So interpretiert die Maschine beispielsweise die binären Bitmuster der Maschinenbefehle auf Registerebene und führt sie aus. Ebenso interpretiert ein Interpreter der Ebene 2 ein PASCAL-Programm der Ebene 3.

Uns interessiert besonders der Übersetzungsvorgang. Es gibt viele Übersetzungsarten – von Compilern bis zu Assemblern. Alle haben jedoch eins gemeinsam: Sie nehmen auf höheren Ebenen geschriebene Befehle entgegen und erzeugen auf einer niedrigeren Ebene den entsprechenden Code. Die ursprünglichen Quellanweisungen werden danach nicht mehr gebraucht und daher vom Übersetzer „weggeworfen" (Interpreter brauchen natürlich immer noch den ursprünglichen Programmtext).

Ziel eines Assemblers ist es, die Assemblerbefehle einer hohen Ebene in den Binärcode zu übersetzen, den die Maschine ausführen kann. Hier ein Beispiel:

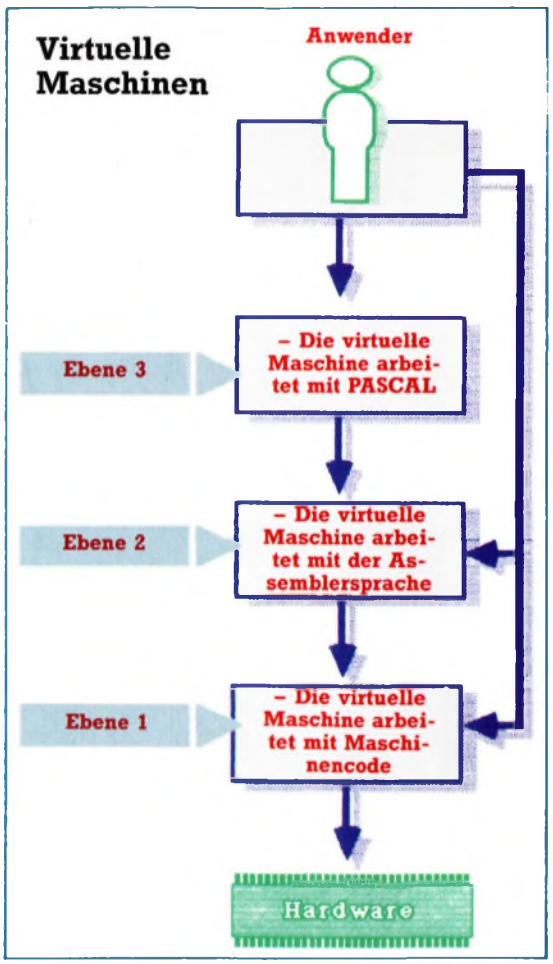
MOVE.W D3,D5
wird übersetzt in:
0011 101 000 000 011

0011 entspricht „Wort übertragen"; 101 000 entspricht „nach D5" und 000 011 „von D3".

Die Bitcodierung ist jedoch viel zu umständlich und fehleranfällig. Wir brauchen zumindest mnemotische Kürzel, um uns Befehle und Datenobjekte merken zu können. MOVE.W TOTAL,D4 sind symbolische Namen, die etwas über den Inhalt einer Speicherstelle aussagen (in diesem Beispiel TOTAL).

Bei der bitweisen Codierung entstehen auch leicht folgende Fehler:

- Befehle (Opcodes) falsch eingegeben
- Falsche absolute Adressierung



Ein Computer kann drei „virtuelle" Maschinen enthalten. Hier läuft auf Ebene 3 die Programmiersprache PASCAL, auf Ebene 2 der Assembler und auf Ebene 1 der Maschinencode, der die Hardware direkt steuert.

● Zuteilung einer falschen Bytezahl durch einen Befehl.

Da die manuelle Übersetzung von mehr als einer halben Befehlsseite kaum mehr zu bewältigen ist, bietet der Assembler mit seiner Übersetzung des Quellentextes in den Objectcode eine unschätzbare Hilfe.

Der Assemblervorgang liest zuerst die Befehle der Quelltextdatei (als ASCII-Text geschrieben) und legt eine Objektdatei an (oder liest sie ein). Anhand eines kurzen Assemblerprogramms (das eine arithmetische Berechnung mit den Elementen eines Arrays durchführt) wollen wir die einzelnen Übersetzungsschritte verfolgen.

Fehlermeldungen

In der ersten Spalte des Programmlistings stehen die Speicherstellen (LOC – siehe Überschriftzeile), gefolgt vom Inhalt der Speicherstellen. Der Rest der Zeile enthält den Quellentext mit den vom Assembler eingesetzten Befehlsnummern. Alle Fehlermeldungen beziehen sich auf diese Nummern. Wenn etwa Zeile 14 einen Fehler enthält, dann erscheint hier ein E, in der Fehlerliste eine Meldung.

Aufbau eines Assemblerlistings

```
LOC       OBJECT                    STMT          SOURCE STATEMENT
                                    1   *This program takes each element of an array and converts
                                    2   *each element to
                                    3   *                INPUT[I]:=2*INPUT[I]+3
                                    4
                                    5
001000                              6         ORG $1000
          =000C                     7   length:   equ 12
                                    8
001000    700C                      9   start:    moveq     #length,d0
001002    41F8  1024                10             lea       input,a0
001006    4244                      11             clr       d4
001008    4243                      12             clr       d3
                                    13
00100A    3610                      14  loop:      move.w    (a0)+,d3
00100C    C7FC  0002                15             muls      #2,d3
001010    D67C  0003                16             add.w     #3,d3
001014    30C3                      17             move      d3,(a0)+
001016    D843                      18             add       d3,d4
                                    19
001018    5340                      20             subq      #1,d0
00101A    6600  FFEE                21             bne       loop
00101E    31C4  103C                22             move      d4,sum
001022    4F40                      23             trap      #0
                                    24
001024    0001  0002  0003  0004    25  input:     dc.w      1,2,3,4,5
          0005
00102F    0006  0007  0008  0009    26             dc.w      6,7,8,9,10
          000A
001038    000B 000C                 27             dc.w      11,12
                                    28
00103C    0000                      29  sum:       dc.w      0
                                    30
                                    31             end

No errors found in this assembly

          SYMBOL                    DEFN          VALUE     REFERENCES
INPUT                               25            1024      10
LENGTH                              7             000C      9
LOOP                                14            100A      21
START                               9U            1000
SUM                                 29            103C      22
```

Auch die Binärdatei ist interessant. Sie enthält außer codierten Binärdaten auch Ladeinformationen, um die Daten überhaupt in den Speicher laden zu können. Da Adreß- und Inhaltsinformation in hexadezimal codierten Binärzahlen gespeichert werden, muß das Lademodul auch das Binärformat verarbeiten können. Zwar bestimmt jeder Assembler das Format selbst, doch ist die entsprechende Kompatibilität des Lademoduls eine der Grundvoraussetzungen für das Funktionieren.

Der letzte Teil des Listings ist die Symboltabelle mit den numerischen Werten, die den (im Programm deklarierten) Labels zugeordnet wurden. So hat INPUT (in Anweisung 25 definiert) beispielsweise den Wert 1024 (hex) und wird in Anweisung 10 angesprochen. In unserem kurzen Beispielprogramm lassen sich diese Daten zwar noch leicht dem Listing entnehmen, bei umfangreichen Programmen sind sie jedoch für die Fehlersuche unentbehrlich.

Die Hardware bestimmt, wohin der übersetzte Text ausgegeben wird. Wenn Sie beispielsweise auf einem Unix-System mit einem „Crossassembler" arbeiten, erscheinen Listing und Binärinformationen in Dateien. Vor Aufruf des Programms müssen die Binärdateien erst in das System des angesprochenen 68000-Prozessors geladen werden.

Ein Assembler kann natürlich auch gleich auf dem gewünschten Prozessor laufen, wenn dort ein Dateisystem vorhanden ist. Dabei werden die Binärcodes direkt in den Speicher geladen und dort assembliert, während ein Drukker das Listing direkt ausgibt.

Bei Einsatz des Assemblers können aber auch Schwierigkeiten auftreten. Normalerweise läuft der Assembliervorgang in zwei ‚Durchgängen' ab. Der erste Durchgang

● decodiert die mnemotischen Kürzel (MOVE, ADD, MULS etc.).
● zählt die Adreßbytes (und bereitet damit die Adreßvergabe vor).
● legt eine Symboltabelle an (die Auskunft über die Symbolwerte gibt).
● meldet Syntaxfehler (ADDQQ ist beispielsweise ein illegaler Befehl).

„Schnellübertragung"

Wenn der Assembler beim ersten Durchgang durch den Quellentext die Anweisung 9 liest, enthält das Label den Wert 41F8. Dies entspricht einer ‚Schnellübertragung' der Konstanten 12 (das Symbol LENGTH) auf DO im unmittelbaren Modus. Die Anweisung 10 (LEA) setzt die Speicherstelle 1002 auf 41F8, kann aber die Adresse von INPUT noch nicht festlegen. Die Speicherstelle 1004 wird daher freigelassen, bis nach dem zweiten Durchgang die Adresse von INPUT bekannt ist.

Im zweiten Durchgang setzt der Assembler alle vom Programm benötigten Adressen ein und gibt den binären Code und das Pro-

grammlisting aus. Fehler kommen dabei nur selten zum Vorschein, da sie zumeist im ersten Durchgang schon abgefangen wurden.

Der Assembler kann auch leicht Berechnungen durchführen und mit symbolischen Ausdrücken den Wert eines Operanden herausfinden. Hier als Beispiel die Einführungszeilen eines Listings:

```
LENGTH DC.W     (START-ARRAY)*4
ARRAY  DC.W     1,2,3,4,5
START  MOVE.W LENGTH,D3
```

Dabei wird die Länge des Arrays in Bytes berechnet (START-ARRAY mal vier) und im Programm eingesetzt (das Datenregister 3 wird gesetzt). Wenn Sie später im Programm die Länge des Arrays ändern wollen, wird LENGTH automatisch neu berechnet.

Auch bei der Programmdokumentation bietet der Assembler praktische Unterstützung. Sie können Kommentare in die Programme einfügen, die die Bedeutung der Befehle erläutern und Auskunft über Registervergabe und Programmaufbau geben. Hier ein Beispiel:

```
* Das Programm nimmt einen Eingabestring ent-
* gegen und prüft ihn.
* Texteingabe und Prüfung per ACIA
* B. Schreiber — Dezember 85
*
* Registervergabe
*
* Adreßregister
*    A1 Pointer auf Eingabestring
*    A2 Pointer auf gespeichertes Schlüsselwort
*    A3 Pointer auf aktuelles Zeichen
* Datenregister
*    D0 Eingabezeichen
*    D1 Ausgabezeichen
*
START JSR INIT          E/A initialisieren
      JSR READSTRING    Eingabestring lesen
      JSR VALIDATE      und prüfen
      JSR SIGNAL        korrekte Eingabe
                        anzeigen
      BRA START         zurück zum Anfang
```

Sie sehen, die Kommentare informieren ausführlich über Programmstruktur und die eingesetzten Register. Auf dieser Ebene wären weitere Kommentare vermutlich überflüssig, doch könnten Sie auf der Subroutinenebene vermutlich weitere Informationen erfahren. Wichtig ist, daß die entsprechenden Programme ausführlich kommentiert sind.

Der Assembler bietet aber noch weitere Hilfen. „Assembleranweisungen" wie TTL veranlassen, daß über jeder Listingseite eine Programmüberschrift gedruckt wird. Andere Anweisungen versorgen den Assembler mit Informationen. So können Sie mit der Anweisung END das Ende des Eingabetextes anzeigen oder mit ORG veranlassen, daß die Assemblierung bei einer bestimmten Adresse anfängt. Ein Programmodul enthält fast immer folgende Zeilen:

```
*    Dokumentationskommentare mit
*    Namen und Datum
     TTL Dies ist mein Programm
     ORG $1000   * Anfangsadresse
START  MOVE D1,D2  * einige Befehle
                   * weitere Befehle
END  * Endanweisung
```

Tabelle der Assembleranweisungen

ORG	Kurzform für ‚Origin'. Legt die Anfangsadresse des Codes fest
RORG	Wie ORG, jedoch für PC relativen Code
TTL	Druckt einen Titel auf jede Seite des Assemblerlistings
END	Kein weiterer Quellentext
EQU	Ordnet dem angegebenen Symbol einen Wert zu (z. B. LENGTH EQU 10+50*ARRAY1. Wenn ARRAY1 12 ist, dann wird LENGTH zu 610)
DC	‚Deklariere Konstante' — weist einen konstanten Wert (Byte, Wort oder Langwort) zu. Z. B. PARAM DC.W 12,23 setzt PARAM als Wort mit Wert 12; PARAM+2 als Wort mit Inhalt 23
DS	Reserviert einen Speicherbereich von nicht initialisierten Daten (z. B. STACK DS.B 100 reserviert einen Adreßbereich von 100 Bytes namens STACK.)
LLEN	Setzt die Zeilenlänge
PAGE	Gibt während des Listings einen Seitenvorschub aus

Gemeinsamkeiten

Die Assembler verschiedener Hersteller unterscheiden sich natürlich in Format und Abläufen voneinander. Dennoch folgen sie meistens dem folgenden Quelltextformat: Kommentare werden durch einen * am Zeilenanfang oder durch mindestens ein Leerzeichen hinter einem Befehl eingeleitet. Folgende Kommentare sind legal:

```
* Dies ist ein Kommentar
START ADD D1,D2 * und so weiter
```

Auch das Befehlsformat ist definiert. Es kann drei Felder enthalten:

- Label: Jeder als Label eingesetzte Name muß mit einem Buchstaben anfangen und insgesamt weniger als 30 alphanumerische Zeichen lang sein. Wenn das Labelfeld leer ist, muß an seiner Stelle mindestens ein Leerzeichen stehen.
- Opcode: Hier stehen die Befehlscodes des 68000.
- Operand: Zwischen Befehlscode und Operanden muß mindestens ein Leerzeichen stehen. Zwei Operanden dürfen nicht durch Leerzeichen voneinander getrennt sein.

Hier einige illegale Anweisungsbeispiele:

```
Dies ist kein Kommentar, da der * fehlt
MOVE  D1, D2     * Leerzeichen zwischen
                 * zwei Operanden
```

```
MOVE   8TOIT,D5  * Namen müssen mit einem
                 * Buchstaben anfangen und
                 * hinter den Operanden
                 * muß mindestens ein
                 * Leerzeichen stehen
RTS              * ist in Ordnung
RTS    D1        * D1 ist hier nicht möglich
* Ende der Fehlerbeispiele
```

Zum Schluß noch ein Wort über die alphanumerische Darstellung. Zahlen werden dem Assembler als Dezimalwerte übergeben, wenn davor kein $-Zeichen steht. So wird 1234 dezimal interpretiert und $1234 als Hexzahl. ASCII-Buchstaben müssen in Anführungsstriche eingeschlossen sein:

```
'Hier ist das Ende des Kurses!'
```

Sie sehen, der Assembler kann nicht nur Code zum 68000 schicken, sondern bietet mit dem Aufbau von Macros, den Assembleranweisungen und Fehlermeldungen auch viele Programmierhilfen. Auch den möglichen Einsatz von Kommentaren sollten Sie nicht unterschätzen, da Sie sich nach mehreren Monaten mit Sicherheit nicht mehr an alle Einzelheiten eines Programms erinnern können.

Der QL arbeitet mit einem 68008 Microprozessor, der dem 68000 entspricht, aber nur einen Acht-Bit-Datenbus besitzt. Da die beiden Chips den gleichen Befehlssatz einsetzen, können Sie die Befehle dieser Serie auch mit dem 68008 verwenden.

Flexibilität mit Macros

Für die Entwicklung umfangreicher Programme können im Quellprogramm, statt umfangreicher Codezeilen, „Macros" angegeben werden. Wenn die Assemblierung auf ein Macro trifft, wird der zuvor definierte Macrotext ausgegeben. Das Macro „ERROR" kann beispielsweise folgendermaßen definiert sein:

```
ERROR   MACRO
        MOVE.B    #3,ERRORLOC
        JSR       SIGNAL
        ENDMACRO
```

Zum Zeitpunkt der Assemblierung erzeugt das Wort ERROR nun folgende Befehle:

```
MOVE.B    #3,ERROR.LOC
JSR       SIGNAL
```

Sie können Macros – falls nötig – auch mit Parametern einsetzen. Hier ein Beispiel:

```
                TTL Macro Example
                ORG       $1000

ADDON   MACRO
        ADD       \1,D2
        ADD       D2,\2
        ENDM

        ADDON     AVAL,SUM
        ADDON     BVAL,SUM
        ADDON     SUM,TOTAL

AVAL    DC.W      0
BVAL    DC.W      0
SUM     DC.W      0
TOTAL   DC.W      0

        END
```

In diesem Beispiel beziehen sich \1 und \2 auf den ersten und zweiten Parameter. Bei der Assemblierung entsteht daraus folgender Code:

```
3                 ORG       $1000
4
5       ADDON     MACRO
6                 ADD       \1,D2
7                 ADD       D2,\2
8                 ENDM
9
10                ADDON     AVAL.SUM
11+               ADD       AVAL,D2
12+               ADD       D2,SUM
13                ADDON     BVAL,SUM
14+               ADD       BVAL,D2
15+               ADD       D2,SUM
16                ADDON     SUM,TOTAL
17+               ADD       SUM,D2
18+               ADD       D2,TOTAL
19
20      AVAL      DC.W      0
21      BVAL      DC.W      0
22      SUM       DC.W      0
23      TOTAL     DC.W      0
24                END
```

Das + hinter den Zeilennummern gibt die Zeilen an, die vom Macro-Prozessor eingesetzt wurden.

Auftrag verstanden, Programm läuft!

Jede neue Aufgabe, die ein Microrechner bewältigen soll, erfordert ein gut abgestimmtes Zusammenspiel zwischen Mensch und Computer.

Wie ein Schallplattenspieler, auf dessen Plattenteller sich keine Platte dreht, ist auch ein Computer nichts anderes als eine Ansammlung aus Metall, Plastik und Silizium, solange in seinem Arbeitsspeicher kein Programm vorhanden ist. Der Computer kann nur dann eine spezielle Aufgabe erfüllen, wenn er „programmiert" wurde.

Selbst ein Laie mit nur wenig Programmiererfahrung wird schon bald zwei völlig unterschiedliche Phasen bei einer Problemlösung erkennen. Das Problem muß zunächst in einer Sprache, die ein Computer verstehen kann, erfaßt und niedergeschrieben werden. Dann erst kann man dieses Programm in den Computer eingeben und testen. Diese beiden Phasen können nun wiederum in zwei Teile unterteilt werden. Hierbei handhabt der Programmierer die erste Phase, während gleichzeitig die zweite Phase im Computer abläuft, ohne daß der Programmierer davon etwas merkt oder etwa eingreift.

Das Problem zerlegen

Angenommen, ein Lohn- und Gehaltsprogramm soll geschrieben werden. Dazu muß zunächst einmal die Problemstellung definiert werden. Was soll der Computer liefern? Welche Informationen benötigt er, um die wöchentlichen Lohnabrechnungen erstellen zu können? In diesen Informationen können z. B. Lohnsätze, wöchentliche Arbeitsstunden etc. enthalten sein. Im nächsten Schritt muß dann die Berechnungsgrundlage für z. B. Steuern und Rentenbeiträge erstellt werden.

Bei einer komplexen kommerziellen Anwendung kann diese Phase durchaus schon von einem „Systemanalytiker" durchgeführt werden, der speziell dafür ausgebildet wurde, Abläufe in einem kaufmännischen Betrieb zu analysieren und in einer Form niederzuschreiben, die sich leicht in ein Programm übersetzen läßt. Für Programme auf Heimcomputern oder für Lehrprogramme wird dieser Schritt jedoch von Programmierern selbst durchgeführt.

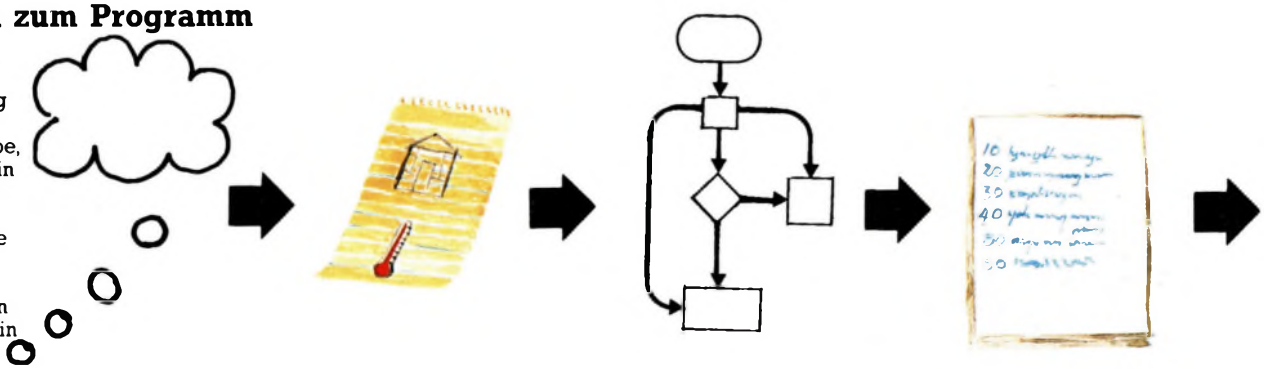
Erfahrene Programmierer zerlegen in dieser Phase das Problem jedoch in weitere kleine Einheiten. Ein Lohnprogramm könnte möglicherweise in vier „Module" zerlegt werden: Das Eingeben der wöchentlichen Daten, die Berechnung der Daten, das Speichern der akkumulierten Ergebnisse, wie z. B. die über das Jahr gezahlten Steuern, und letztlich den Ausdruck der Lohnabrechnung.

Jedes Hauptmodul wird dann wiederum in noch kleinere Strukturen unterteilt. Dieser Vorgang ist als „Strukturierte Programmierung" bekannt. Jede dieser kleinen Strukturen ist übersichtlich und kann in dem Programm mit einer oder zwei Zeilen verwirklicht werden. Schließlich werden alle Programmzeilen in den Computer eingegeben.

Von dem Augenblick an, in dem RUN eingegeben wird, übernimmt der Computer die Steuerung, und wieder sind in diesem Stadium

Vom Problem zum Programm

Ein Programm wird erstellt, wenn für ein Problem eine Lösung gefunden werden muß, z. B. die Aufgabe, wie die Temperatur in einem Gewächshaus konstant gehalten werden kann. Um die Antwort zu finden, muß das Problem verschiedene Stadien durchlaufen, bevor ein vollständiges Programm entsteht.



Ein Problem taucht auf . . .

Es wird auf einem Stück Papier in groben Zügen dargestellt.

Eine Ablaufskizze wird erstellt, um das Problem zu analysieren und die Struktur des Programms zu entwickeln . . .

Die Ablaufskizze wird in eine der Computersprachen übersetzt, z. B. in BASIC.

viele unterschiedliche Ebenen an dem Ablauf beteiligt. Diese Abläufe innerhalb des Computers bleiben jedoch dem Anwender verborgen. Er merkt lediglich, daß der Computer ihn nach den entsprechenden Eingaben fragt und dann die gewünschten Ergebnisse produziert.

Da Microprozessoren eine Hochsprache wie BASIC nicht unmittelbar verstehen, müssen alle Befehle in Maschinencode übersetzt werden. In den Heimcomputern wird diese Aufgabe von dem „Interpreter" (einem Übersetzungsprogramm) übernommen, das fest im ROM des Computers installiert ist.

Der Interpreter ist ein hochentwickeltes Programm im Maschinencode und wird direkt von dem Microprozessor ausgeführt. Wird der Befehl RUN eingegeben, untersucht der Interpreter zunächst das von dem Anwender eingegebene Programm Zeichen für Zeichen und vergleicht alle Befehle, die er findet, mit seinem eigenen internen „Wörterbuch". Findet er ein Zeichen, das er nicht versteht, da es durch einen Eingabefehler erzeugt wurde, beendet er die Übersetzung des Programms und gibt die Meldung „SYNTAX ERROR" (Schreibfehler) auf den Bildschirm aus.

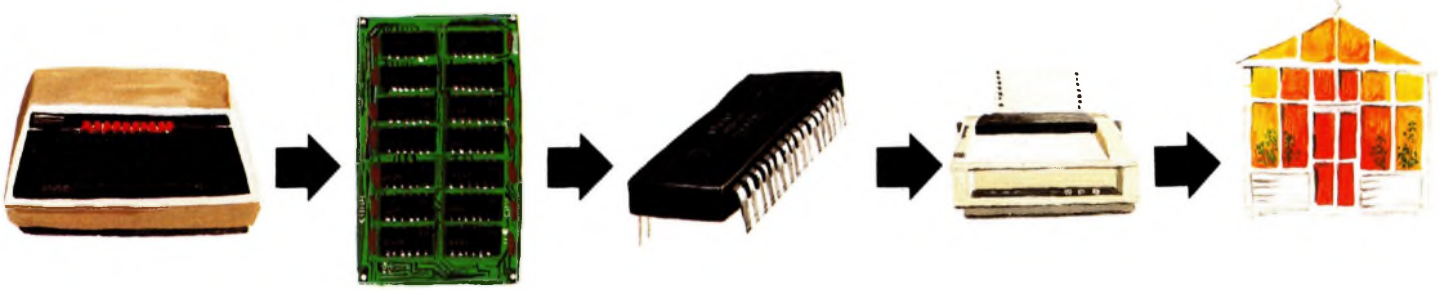
Ist ein Wort aber im „Wörterbuch" des Interpreters vorhanden (z. B. PRINT), übermittelt er diesen Befehl direkt an den Teil des Interpreters, der weiß, wie diese Funktion ausgeführt wird. In diesem Fall wird dieser Teil des Interpreters untersuchen, welche Zeichen auf das Wort PRINT folgen und die Daten zur Darstellung vorbereiten.

An dieser Stelle tritt eine weitere Ebene des Computers in Aktion. Im Arbeitsspeicher des Computers existiert eine Routine, die eine Folge von Zeichen aufnehmen und in einem speziell für den Bildschirm reservierten Speicherbereich unterbringen kann. Diese Routine steuert auch die Umwandlung der Zeichen in Impulse, die eine Darstellung auf dem Bildschirm oder Monitor erst ermöglichen. Auch wenn das Programm mit anderen Arbeiten beschäftigt ist, läuft dieser Vorgang parallel ab.

Mit der Tastatur des Computers findet ein ähnlicher Vorgang statt. Ständig prüft ein besonderes Programm, ob Tasten gedrückt werden und schickt diese Zeichen im Fall einer Eingabe in den Speicherbereich, der für die Eingabeinformationen des laufenden Programmes reserviert ist. Damit auch die Möglichkeit besteht, ein Programm im Ablauf jederzeit mit der BREAK-Taste zu unterbrechen, muß ständig eine Überprüfung der Tastatur stattfinden, selbst wenn ein Programm läuft.

Es müssen also auf vielen Ebenen Abläufe stattfinden, damit der Computer Ergebnisse liefern kann – selbst wenn ein Programm bereits eingegeben ist. Diese Vorgänge mögen sehr kompliziert erscheinen, sie werden jedoch fast alle vom Computer ohne einen Eingriff von außen gesteuert.

In jedem Computer existiert eine komplexe Programmhierarchie. Bei ihren vielen Aufgaben überprüft sie unter anderem, was auf dem Bildschirm dargestellt ist, ob und welche Taste auf der Tastatur gedrückt wurde, welche Befehle an die Peripheriegeräte ausgegeben wurden, und in welchem Zustand sich der Inhalt des Arbeitsspeichers befindet. Alle diese Funktionen laufen ständig ab, während sich der Anwender nur um sein nächstes Programm zu kümmern braucht.



Das Programm wird über die Tastatur in den Arbeitsspeicher des Computers eingegeben.

Das BASIC-Programm wird durch eine Reihe von Chips geleitet, die es in Maschinencode umwandeln.

Programm-Prozesse werden in der CPU ausgeführt und an ein Peripheriegerät übermittelt.

In diesem Fall stellt der Drucker einen Ausdruck (oder „Hardcopy") her.

Funktioniert das Programm fehlerfrei, ist das Problem gelöst.

Dolmetscher für Maschinensprache

Zwei unterschiedliche Umwandlungsprogramme, Interpreter und Compiler, sorgen für die Übersetzung von Eingaben und Befehlen in die für Computer verständliche Maschinensprache.

Die ersten Computer besaßen noch keine Tastaturen. Programmbefehle mußten mühsam einzeln über eine Leiste mit acht Schaltern eingegeben werden, die entweder auf „an" oder „aus" gestellt werden konnten und damit einen bestimmten Befehl darstellten. Die Schaltmuster waren die ersten Entsprechungen des Maschinencodes.

In der logischen Weiterentwicklung ersetzte eine Schreibmaschinentastatur die Schaltleiste und englische Begriffe die ursprünglichen Schaltmuster. Am Ende entwickelten sich Hochsprachen wie BASIC, die das Programmieren in Maschinensprache verdrängten.

Der Computer veränderte sein Funktionsprinzip jedoch nicht, und noch heute arbeiten Prozessoren auf der Grundlage der alten Schaltmuster. Damit der Prozessor aber Programmbefehle einer Hochsprache ausführen kann, mußten in Maschinencode Übersetzungsprogramme geschrieben werden, die die Hochsprachen in eine für den Prozessor verständliche Form umwandelten. Diese Umwandlungsprogramme werden je nach Übersetzungsmethode Compiler oder Interpreter genannt.

Interpreter werden in Heimcomputern normalerweise gleich mitgeliefert und sind eine Methode, Hochsprachen in einen kostengünstigen Code zu übersetzen, den der Computer verstehen kann. Sie benötigen wenig Speicherplatz und stellen daher viel Speicher für Programme und Daten zur Verfügung.

Microcomputer, die weniger als 1600 Mark kosten, besitzen fast immer einen BASIC-Interpreter. Man gibt ein BASIC-Programm ein, tippt RUN und das Programm funktioniert – oder bricht mit einer Fehlermeldung des Systems ab wie z. B.:

```
SYNTAX ERROR ON LINE 123
```

Wenn man dann LIST eingibt, den Fehler korrigiert und wiederum RUN tippt, läuft das Programm oder stoppt bei dem nächsten Fehler usw. Inzwischen gibt es BASIC-Interpreter, die Fehler bereits bei der Eingabe von Programmzeilen anzeigen.

Zwar ist der Interpreter leicht anzuwenden, er weist aber einige Nachteile auf: Jedesmal, wenn Sie RUN eingeben, muß er das Programm im Speicher finden, es Zeile für Zeile übersetzen und dann ausführen. Befindet sich in Ihrem Programm folgende Schleife

```
400 LET N=0
500 PRINT N
600 LET N=N+1
700 IF N<100 THEN GOTO 500
```

so muß der Interpreter die Zeilen 500 bis 700 hundertmal von neuem übersetzen und ausführen, als ob er nie vorher auf sie gestoßen wäre.

Compiler funktionieren nach einem völlig anderen Prinzip. Sie sind teuer, recht kompliziert und benötigen für sich und ihre Programme viel Speicherplatz. Um arbeiten zu können, brauchen sie fast immer Diskettenlaufwerke als Zwischenspeicher.

Compiler bieten auf der anderen Seite enorme Flexibilität und Geschwindigkeit. Unsere vier BASIC-Zeilen oben würden nur ein einziges Mal übersetzt werden, der entsprechende Code jedoch einhundertmal ausgeführt.

Dieser Zeitgewinn hat seinen Preis. Nehmen wir an, Sie haben einen BASIC-Compiler und möchten ein BASIC-Programm ausführen. Dafür müssen Sie zunächst den Editor laden und starten. Mit diesem Editor schreiben Sie das Programm und speichern es als sogenannten „Quellcode" auf eine Diskette.

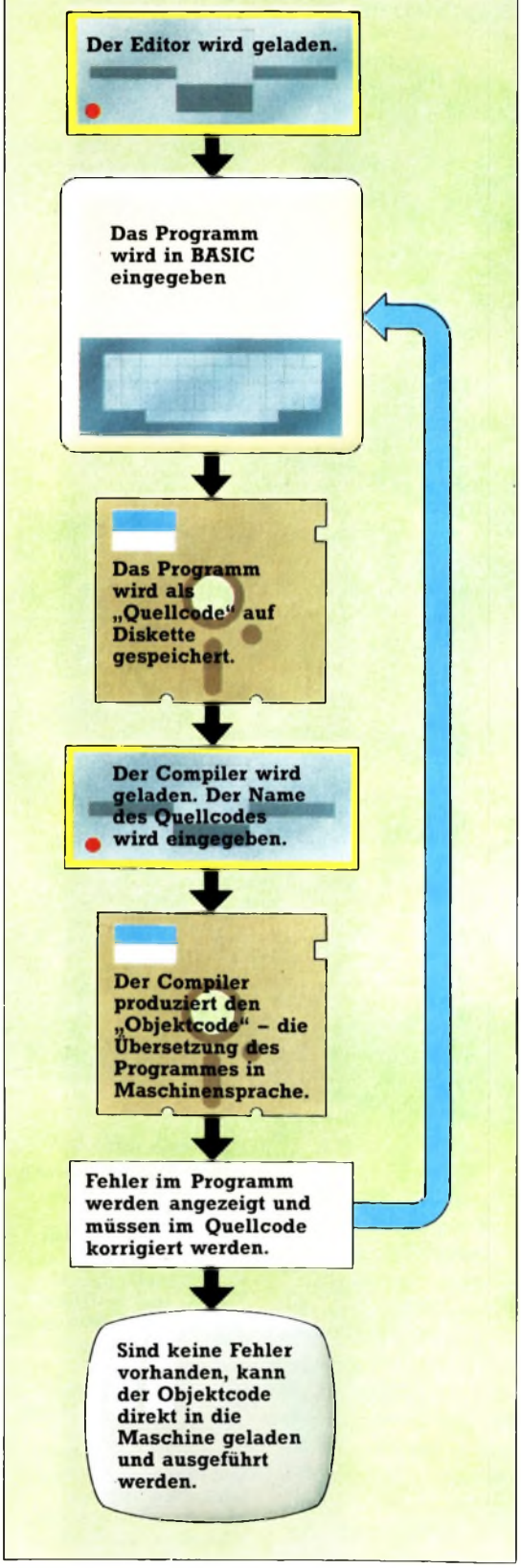
Programme müssen Namen haben, damit man sie im Speicher wiederfinden kann. Der Editor fragt daher zunächst nach dem Name des Quellprogramms. Programmnamen bestehen oft aus zwei Teilen: Der erste Teil ist die Bezeichnung wie z. B. TESTPROG – und der zweite Teil ist normalerweise ein Code aus drei Zeichen, der den Programm- oder Dateityp bezeichnet. Dieser Teil des Namens wird „Extension" oder „Kürzel" genannt. Ein BASIC-Programm mag den Code BAS als Kürzel haben, und Ihr Quellcode mag daher auf der Diskette unter dem Namen TESTPROG. BAS abgelegt sein. Jetzt geben Sie ein:

```
COMPILE TESTPROG. BAS
```

und der Computer lädt den Compiler, startet

Ein Programm compilieren

Ein Programm selbst zu compilieren ist weitaus schwieriger als die Verwendung eines Interpreters. Ein solches Programm läuft jedoch schneller ab. Dies sind die wesentlichen Schritte zur Erstellung eines compilierten Programms:



ihn und läßt ihn den Quellcode TESTPROG. BAS in Maschinencode übersetzen.

Nach kurzer Zeit hat der Computer Ihr Programm in einen „Object Code" umgewandelt, den er unter dem Namen TESTPROG. OBJ auf der Diskette speichert. Das Kürzel OBJ zeigt an, daß sich hier ein in den Maschinencode übersetztes Quellprogramm befindet.

Während der Compiler das Quellprogramm übersetzt, überprüft er es auf Eingabefehler. Findet er Fehler, zeigt er sie in folgender Form an:

```
100 REED X:IF X=3(N+2) LET P=Q
      1          2           3
FATAL ERROR:—
1) //REED// UNRECOGNISED COMMAND
2) ///(//ILLEGAL OPERATOR HERE
3) ??'THEN' OR 'GOTO' EXPECTED HERE
```

Für jede Programmzeile, die einen Fehler enthält, wird eine derartige Nachricht auf den Bildschirm gebracht. Die Fehlermeldungen sind weitaus ausführlicher als bei einem BASIC-Interpreter. Ist ein Fehler aufgetreten, muß der Editor neu geladen und gestartet, der Quellcode neu von der Diskette eingelesen und korrigiert werden. Dann beginnt der Vorgang der Compilierung von neuem.

Sind keine weiteren Eingabefehler vorhanden, können Sie

```
RUN TESTPROG
```

eingeben, und das Programm funktioniert entweder, wie Sie es sich gedacht haben, oder es müssen noch weitere Änderungen vorgenommen werden.

Die Vorteile eines Compilers treten im Entwicklungsstadium eines Programmes nicht besonders zutage, obwohl die ausführlichen Fehlermeldungen eine große Programmierhilfe sein können. Compiler zeigen ihre Stärke erst, wenn ein Programm fertig ist und Sie RUN eingegeben haben.

Compilierte Programme sind bis zu 50 mal schneller als interpretierte Programme. Die Schnelligkeit eines compilierten Programmes wird jedoch durch mühselige und langsame Programmentwicklung erkauft.

Ist ein Programm erst einmal compiliert, so wird das Quellprogramm nur noch als Referenz benötigt und kann daher lesbar und mit ausführlichen Kommentaren versehen programmiert werden. Das compilierte Programm kann dabei weitaus kürzer werden als das Quellprogramm, da Kommentare nicht mit in den Maschinencode übersetzt werden.

Die Tatsache, daß Objektcode aus unverständlichem Maschinencode besteht, kann auch von Vorteil sein. Softwarehäuser verkaufen allgemein nur Programme im Objectcode, der weitaus schwerer zu ändern ist und dessen Sicherungen kaum gebrochen werden können.

Für ein Abenteuerspiel ist Geschwindigkeit nicht wichtig, da der Großteil des Programms aus Texten besteht. Sie sind oft in BASIC geschrieben und werden während des Ablaufs übersetzt.

Weil sie sehr viele Berechnungen enthalten, können zahlreiche kommerzielle Programme (besonders Kalkulationsprogramme) nur unter großen Schwierigkeiten in Maschinencode programmiert werden.

Für schnelle Weltraumspiele mit sich ständig verändernder Grafik sind sogar compilierte Programme nicht schnell genug. Sie müssen direkt in Maschinencode programmiert werden.

Bauen Sie den „Turm von Hanoi"!

Mehr über recursive Prozeduren, eigene Hilfsprogramme und die Bedingungsüberprüfung auf TRUE oder FALSE.

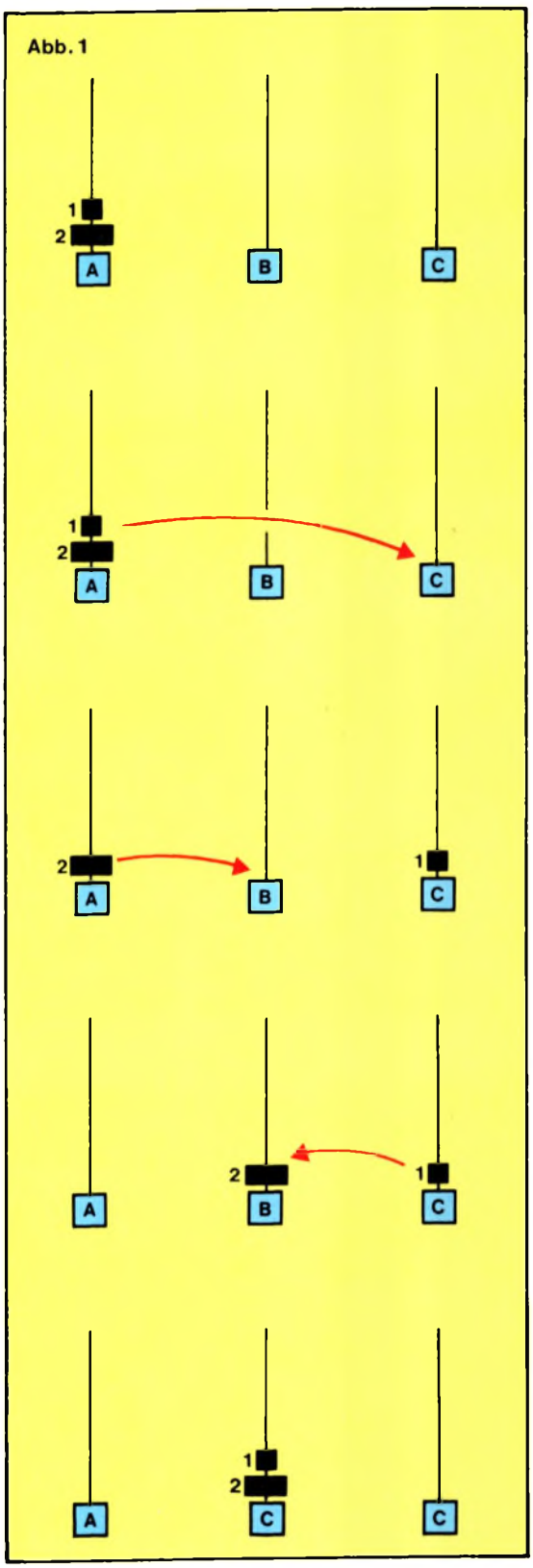
Ein gutes Beispiel für recursive Prozeduren ist ein Sortierprogramm, das Gegenstände nach unterschiedlichen vorgegebenen Kriterien ordnet. Die Bestandteile dieses „Puzzles" setzen sich aus mehreren Scheiben verschiedener Größe zusammen. Diese sind zu Beginn auf einem der drei zur Verfügung stehenden Holzstifte aufgesetzt, und zwar so, daß die größte Scheibe unten liegt, darauf befindet sich die nächstkleinere usw. Die kleinste Scheibe liegt obenauf.

Das Programm, wir nennen es „Turm von Hanoi", soll die Scheiben von einem Stift auf einen anderen „umschichten". Um die korrekte Reihenfolge zu gewährleisten, können die Scheiben nur einzeln transportiert werden.

Bevor die Prozedur geschrieben wird, sollten die jeweiligen Schritte sorgfältig durchdacht und zur Veranschaulichung auf Papier gezeichnet werden. Die erste Abbildung zeigt den Sortiervorgang mit zwei Scheiben. Das erscheint noch relativ einfach. Die Aufgabenstellung sieht folgendermaßen aus: Die beiden Scheiben (1 und 2) sollen von Stift A auf B gesetzt werden, wobei die ursprüngliche Reihenfolge beibehalten werden muß. Um nun die größere Scheibe (2) als erstes Element auf Stift B legen zu können, muß zunächst Scheibe 1 entfernt, das heißt, auf Stift C gelegt werden. Danach wird zunächst Scheibe 2 und anschließend Scheibe 1 auf Stift B gesetzt. Fertig!

Das Prinzip dieses Sortiervorgangs sollte nun klar sein. Schwieriger wird es jedoch, wenn sechs Scheiben umgeschichtet und sortiert werden müssen (Abbildung 2). Dazu muß wiederum Scheibe 6 (die größte) von Stift A auf B gelegt werden. Was bedeutet, daß die ersten fünf Scheiben von Stift A auf C zu stecken sind. Diese Aufgabe wird später ein Unterprogramm in der Prozedur lösen. Es entsteht also ein neuer „Turm von Hanoi" aus fünf Elementen. Mit der nachfolgenden recursiven Prozedur lassen sich die Scheiben korrekt ordnen:

```
TO HANOI :NUMMER :VON : NACH :TEMP
  IF :NUMMER = 0 [STOP]
  HANOI :NUMMER — 1 :VON :TEMP :NACH
```

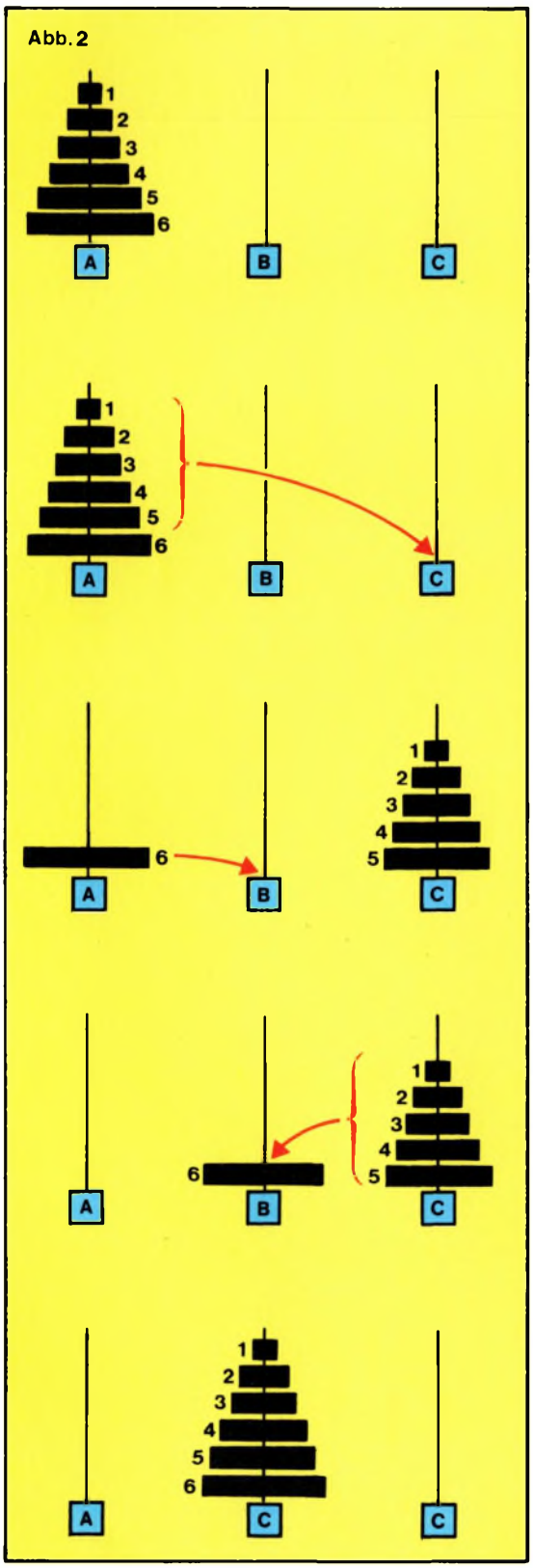


Die beiden Scheiben befinden sich zunächst auf Stift A. Es soll erreicht werden, daß die Scheiben wiederum nach Größe geordnet auf Stift B gesetzt werden.

```
  PR (SENTENCE [MOVE DISK] :NUMMER [VON
     STIFT] :VON [NACH STIFT] :NACH)
  HANOI :NUMMER — 1 :TEMP :NACH :VON
END

HANOI 6 "A "B "C
```

Jetzt wird's schon schwieriger. In diesem Beispiel sind sechs Scheiben geordnet auf einen anderen Stift zu legen.



Bei diesem Programm müssen vier Eingaben eingetippt werden. Mit der Variablen :NUMMER wird festgelegt, wieviel Elemente die Prozedur sortieren soll. Die Angaben "A, "B und "C bezeichnen die drei Holzstifte, auf die die geordneten Scheiben aufgesteckt werden.

Drei Variationen

Zur Übung sollten Sie nun versuchen, diese Problemstellung in einer anderen Programmiersprache zu lösen. Sie werden sehen, daß das Programm erheblich länger wird, als die LOGO-Version. Das nächste Beispiel für recursive Prozeduren entstand aus der Zusammenarbeit einiger amerikanischer High-School-Studenten und LOGO-User-Gruppen. Die Überlegung dabei war, Geometrie anhand von selbständigen Experimenten und Entwicklungen zu vermitteln. Dabei sollten natürlich auch die Vorurteile derjenigen abgebaut werden, die in Turtle Graphics nichts anderes als ein nettes Spielzeug sehen. Die folgende Prozedur gibt eine vorgegebene Zeichnung in drei verschiedenen Variationen aus:

```
TO VERZWEIGEN :LAENGE :STUFE
   IF :STUFE = 0 [STOP]
   FORWARD : LAENGE
   LEFT 45
   VERZWEIGEN :LAENGE / 2 :STUFE — 1
   RIGHT 90
   VERZWEIGEN :LAENGE / 2 :STUFE — 1
   LEFT 45
   BACK :LAENGE
END
```

Die Befehle, die in diesem Programm verwendet werden, sind bereits vorher ausführlich besprochen worden. Zur Erinnerung noch einmal der Programmablauf in Kurzform. Die Variable :LAENGE steht für die Strecke, die bei jedem Programmdurchgang von der Turtle zurückgelegt wird, und :STUFE repräsentiert den Wert, der angibt, wie oft die recursive Prozedur abgerufen werden soll.

Ändern Sie nun das Programm so, daß auch verschiedene Variablenwerte und andere Bewegungsabläufe als Eingaben möglich sind. Oder Sie versuchen, eine Prozedur – unter Verwendung der oben gezeigten – zu schreiben, die mehrere Bäume auf den Bildschirm zeichnet. Abbildung 3 zeigt ein Beispiel, wie dieser Baum aussehen könnte.

Hilfsprogramme

Wie bereits erwähnt, bietet LOGO die Möglichkeit, eigene, vielseitig einsetzbare Unterprogramme zu definieren, die je nach Wunsch in die Prozeduren einfügt oder von ihnen aufgerufen werden können. Auf diese Weise lassen sich auch die Möglichkeiten, die Befehle wie IF, THEN oder ELSE bieten, erheblich er-

weitern. Die folgenden Zeilen zeigen Beispiele für selbstdefinierte Hilfsprogramme, in denen ebenfalls Bedingungen abgefragt werden.

Die folgende recursive Prozedur überprüft eine Bedingung und ruft sich selbst solange auf, bis das Ergebnis des Testes „FALSE" ergibt.

```
TO WAEHREND :BEDINGUNG :AKTION
  IF RUN :BEDINGUNG [RUN :AKTION] [STOP]
  WAEHREND :BEDINGUNG :AKTION
END
```

Durch den Befehl RUN kann man die Anweisungen, die in der Liste stehen, aufrufen. Wie sich diese Prozedur in Verbindung mit einer neuen anwenden läßt, demonstrieren die nächsten Zeilen:

```
TO ZAEHLEN
  MAKE "N 1
  WAEHREND [ :N < 20] [PRINT :N MAKE
    "N :N + 1]
END
```

Diese Prozedur „zählt" von 1 bis 20 und gibt die jeweiligen Zahlen auf dem Bildschirm aus.

Die nächste selbstdefinierte Bedingungsüberprüfung nennen wir „VON . . . BIS". Die Prozedur ruft sich auf, bis das Ergebnis der Prüfung, das je nach Vorgabe FALSE oder TRUE lauten soll, korrekt ist.

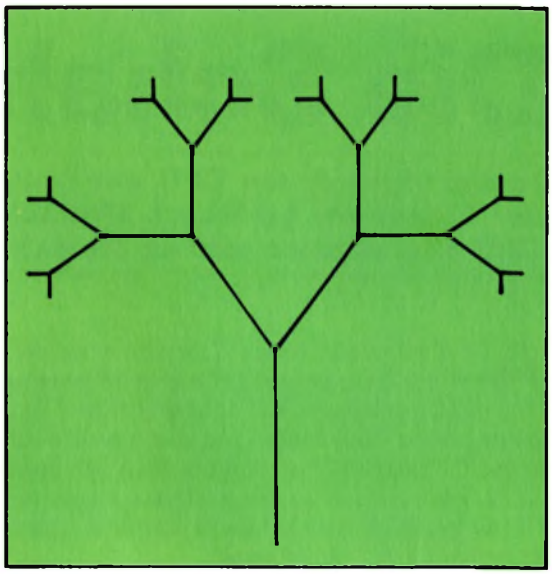
```
TO VON :AKTION :BIS :BEDINGUNG
  RUN :AKTION
  IF BIS :BEDINGUNG [STOP]
  VON :AKTION :BIS :BEDINGUNG
END

TO BIS :BEDINGUNG
  IF RUN :BEDINGUNG [OP "TRUE] [OP "FALSE]
END
```

Die Prozedur VON beinhaltet als Parameter drei eigenständige Listen. Geben Sie nun das folgende Programm ein:

```
TO VERSUCH
  MAKE "N 1
  VON [PRINT :N MAKE "N :N + 1] [BIS]
    [ :N > — 1]
END
```

Die Variable :N wird mit einem Wert initialisiert, der größer ist als derjenige, auf den die Variable überprüft werden soll. Die Prozedur :AKTION wird vor der :BEDINGUNG aufgerufen. Das Programm :BIS dagegen beinhaltet nur eine Liste von Parametern, die die Eingaben auf TRUE oder FALSE überprüfen. Versuchen Sie nun, diese Prozeduren in eigene, von Ihnen selbst entwickelte Programme einzubauen und die Werte entsprechend Ihren Vorgaben zu verändern.



Schreiben Sie eine Prozedur, die unter Verwendung der Recursion mehrere Bäume auf den Bildschirm zeichnet.

Spiel mit Linien

Zur Entspannung hier noch ein kleines Programm, in dem Sie Ihre Kenntnisse der Turtle Graphics erproben können:

```
TO BILD1 :A :B :X1 :Y1 :X2 :Y2
  IF :A = 0 THEN STOP PENUP
  SETXY :X1 :Y1 PENDOWN
  SETXY :X2 :Y2
  MAKE "X1 :X1 + :B
  MAKE "Y2 :Y2 + :B
  BILD1 :A — 1 :B :X1 :Y1 :X2 :Y2
END

TO BILD2 :A :B :X1 :Y1 :X2 :Y2
  IF :A = 0 THEN STOP PENUP
  SETXY :X1 :Y1 PENDOWN
  SETXY :X2 :Y2
  MAKE "X1 :X1 — :B
  MAKE "Y2 :Y2 + :B
  BILD2 :A — 1 :B :X1 :Y1 :X2 :Y2
END

TO DEMO1
  BILD1 25 10 (—158) (—120) 158 (—120)
END

TO DEMO2
  BILD2 25 10 158 (—120) (—158) (—120)
END

TO ZEICHNUNG
  FULLSCREEN DEMO1 DEMO2
END
```

Bei dieser Prozedur stellen X1 und Y1 die Anfangs-, X2 und Y2 die Endkoordinaten der einzelnen Linien dar. Der Abstand zwischen den Linien ist in der Variablen B und die Anzahl der zu zeichnenden Linien in der Variablen A abgelegt. Versuchen Sie nun durch Änderung der Werte neue Zeichnungen zu erstellen.

Welche Sprachen verstehen Computer?

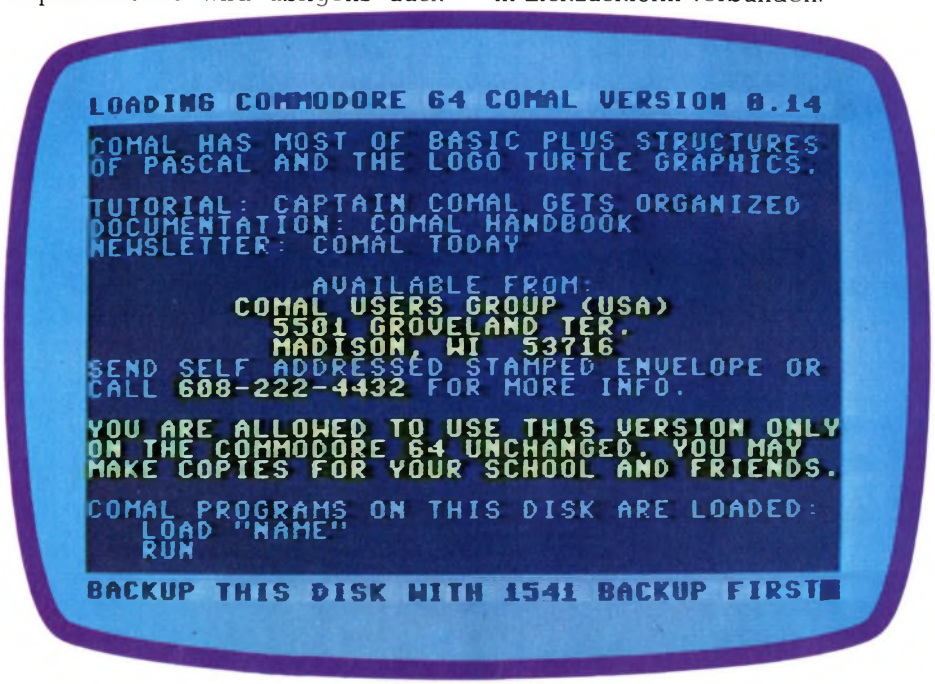
? Wieviele Arten der Datenübertragung gibt es?

Man unterscheidet zwischen drei verschiedenen Arten der Datenübertragung – neben Simplex-Betrieb gibt es Halb- und Duplex-Betrieb. Beim sogenannten Simplex-Betrieb werden die zu übertragenden Daten beispielsweise über einen Übertragungsweg von einem Computer zu einem Peripheriegerät geschickt. Dabei dient ein Gerät als Sender, das andere als Empfänger. Der Datenfluß findet also lediglich in einer Richtung statt. Im Gegensatz dazu steht das Halb-Duplex-Verfahren, wobei die beiden Geräte abwechselnd die Rolle des Empfängers und Senders übernehmen. Anders als beim Simplex-Verfahren wird hier in beiden Richtungen übertragen. Da die Übertragung wechselweise stattfindet, wird dieser Vorgang Wechselbetrieb genannt. Beim Duplex-Verfahren wiederum findet der Datenverkehr auf zwei parallelen Datenkanälen statt. In puncto Schnelligkeit ist dieses Verfahren den beiden anderen, Simplex und Halb-Duplex, klar überlegen. Der Duplex-Betrieb wird übrigens auch Gegen-Betrieb und Voll-Duplex-Betrieb genannt.

? Was ist der Unterschied zwischen Einzelblatt- und Endlospapierzuführung?

Für den Einzelblatteinzug, auch Formulareinzug genannt, ist eine spezielle Mechanik, die auf den Drucker gesetzt wird und ihn mit einzelnen Blättern versorgt, notwendig. Doch nicht jeder Drucker verfügt über die technischen Voraussetzungen, Einzelblatteinzug betreiben zu können. Man unterscheidet ferner zwischen Einzelblatteinzügen für eine oder mehrere Sorten Papier (Einschacht- oder Zweischachtmechanik). Die Hersteller von Druckern bieten Einzelblatteinzüge meist als Zusatzgerät an.

Endlospapier dagegen sind zusammenhängende Papierbögen, die auf beiden Seiten gelocht sind. Das Papier wird über eine mit Stachelwalzen bestückte Traktorführung in den Drucker eingezogen. Die Seiten sind in Zickzackform verbunden.



? Was ist eigentlich ein „Fenster"?

Mit der sogenannten Fenstertechnik werden Programme anwenderfreundlicher gemacht. Vergleichbar mit einzelnen Seiten, die man in einem Hefter (Bildschirm) ablegt, werden in verschiedenen Fenstern verschiedene Programme abgelegt, die dem Benutzer ständig zugänglich sind. Zwischen mehreren Fenstern oder Programmen ist ein Austausch der Daten möglich. Man könnte sich in einem solchen Fall ein Bilanzprogramm vorstellen, das mit Grafik versehen werden soll, um die verschiedenen Zahlengrößen grafisch, beispielsweise als Blöcke, darzustellen. Man unterscheidet zum Beispiel zwischen Text- und Grafikfenstern.

? Warum muß man mit den meisten Computern englisch sprechen?

Die Weltsprache Englisch wird in vielen Ländern gesprochen und in den meisten anderen zumindest verstanden. Sie ist die einzige auf der gesamten Erde gebräuchliche Sprache. Computerhersteller konstruieren die Geräte mit englischen Benennungen für Funktionstasten und Ein- und Ausgänge. Auch Programme werden mit Befehlen gestaltet, die aus dem Englischen stammen. Auf diese Weise wird erreicht, daß nur eine Art von Geräten und Programmen erstellt werden muß, und trotzdem kann der Computer in der ganzen Welt verkauft werden – in Norwegen wie in Japan. Einige Programmiersprachen bieten jedoch die Möglichkeit, bestimmte Befehle in einer anderen Sprache, zum Beispiel in Deutsch, zu definieren.

Auf der ganzen Welt, auch bei Computerbenutzern, hat sich Englisch als Weltsprache durchgesetzt. Die meisten Programme enthalten einheitliche Befehle in englischer Sprache.

Andere Sprachen

Als Nachtrag zu unserem BASIC-Programmierkurs wollen wir einige der Stärken und Schwächen von BASIC im Vergleich zu anderen Programmiersprachen aufzeigen.

BASIC ist eine Weiterentwicklung von FORTRAN, einer der ersten Programiersprachen. Im Gegensatz zu den meisten anderen Sprachen ist BASIC eine Interpretersprache. Das bedeutet, daß bei der Ausführung eines BASIC-Programmes ein anderes spezielles Programm irgendwo im Speicher des Computers den Code Zeile für Zeile interpretiert und die BASIC-Anweisungen in Maschinensprache übersetzt. Betrachten Sie das folgende kurze BASIC-Programm:

```
 10 CLS
 20 PRINT "GEBEN SIE EINE ZAHL EIN"
 30 INPUT X
 40 PRINT "GEBEN SIE EINE ZWEITE ZAHL
    EIN"
 50 INPUT Y
 60 PRINT "DAS PRODUKT DER BEIDEN
    ZAHLEN IST: ";
 70 PRINT X*Y
 80 PRINT
 90 PRINT "WOLLEN SIE EINEN WEITEREN
    DURCHLAUF?"
100 PRINT "DRUECKEN SIE 'J' FUER
    EINEN NEUEN VERSUCH"
110 PRINT "ODER 'N' UM
    AUFZUHOEREN"
120 FOR X=1 TO 1
130 LET A$=INKEY$
140 IF A$ <> "J" AND A$ <> "N"
    THEN X=0
150 NEXT X
160 IF A$="J" THEN GOTO 10
170 END
```

Wenn der BASIC-Interpreter Zeile 10 untersucht, arbeitet er den Maschinencode aus, der zum Löschen des Bildschirmes notwendig ist. Für Zeile 20 ruft er die notwendigen Maschinencode-Anweisungen auf, um die Meldung GEBEN SIE EINE ZAHL EIN auf dem Bildschirm darzustellen. Bei Zeile 30 würde er den Speicherplatz zum Speichern einer realen Zahl reservieren, auf eine Eingabe von der Tastatur warten und dann die eingegeben Zahl in eine Binärzahl umwandeln und an dem für die Variable X reservierten Speicherplatz ablegen. Das ganze wird dann für die Zeilen 40 bis 60 wiederholt. Wenn der Anwender das Programm noch einmal verwenden möchte (indem er J eingibt), würde der Interpreter wieder zu Zeile 10 verzweigen und alle eben genannten Berechnungen erneut ausführen.

Die meisten anderen Sprachen sind „Compilersprachen". Das bedeutet, daß das Programm, nachdem es geschrieben wurde, durch einen sogenannten Compiler bearbeitet werden muß, bevor man es starten kann. Der Compiler ist ein separates Programm, das den „Source Code" durcharbeitet (das Original-Programm) und eine zweite Fassung in Maschinensprache erstellt. Wenn das compilierte Programm gestartet wird, ist es erheblich schneller als ein interpretiertes Programm, da der gesamte zeitaufwendige Prozeß der Übersetzung in Maschinensprache bereits abgeschlossen ist.

Abgesehen von der größeren Schnelligkeit compilierter Programme bieten interpretierte Sprachen einige Vorteile. Diese resultieren daraus, daß interpretierte Sprachen interaktiv sind. Das bedeutet, daß man das Programm während der Entwicklung testen und bearbeiten kann. BASIC gestattet beispielsweise, daß man an jedem beliebigen Punkt eines Programmes eine STOP-Anweisung integrieren kann. Wenn der Interpreter das STOP erreicht, unterbricht er die Interpretierung des Programmes und erlaubt die weitere Eingabe von Befehlen über die Tastatur.

Leicht erlernbar

Befehle sind Anweisungen, die vom Interpreter direkt ausgeführt werden können. Wenn ein BASIC-Programm ausgeführt wurde (wenn z. B. der Interpreter die END-Anweisung erreicht hat) oder wenn der Interpreter eine STOP-Anweisung findet, ist es möglich, alle Variablenwerte auszudrucken.

BASIC wird oft als die ideale Sprache für den unerfahrenen Programmierer bezeichnet, da man Fehler direkt über die Tastatur beheben kann. In unserem BASIC-Programmierkurs haben wir bereits alle fundamentalen und einige der fortgeschrittenen Aspekte von BASIC behandelt. Syntax-Fehler, wie etwa 40 PRNT A(12) ergeben beim Programmlauf normalerweise leicht verständliche Fehlermeldungen, wie SYNTAX ERROR IN 40. Der Hinweis auf die Zeilennummer, in der der Fehler aufgetreten ist, erleichtert das Erkennen des Fehlers. Sobald der Interpreter einen Fehler in der Syntax

oder Logik findet, unterbricht er die Programmausführung und gibt eine Fehlermeldung ab. Die Beseitigung der Ursache ist dank der zahlreichen Editier-Befehle genauso einfach wie die Eingabe einer neuen Programmzeile. Vorausgesetzt, die Fehlerquelle wurde herausgefunden. Nach der Korrektur wird das Programm mit RUN erneut gestartet.

Langsame Verarbeitung

Trotz aller Vorteile hat die Programmiersprache BASIC auch einige Nachteile, geringfügige und schwerwiegende. Da BASIC interpretiert wird (obwohl es auch einige Compiler für BASIC gibt), ist es in der Ausführung sehr langsam. Wenn Geschwindigkeit wichtig ist (wie z. B. bei einem Programm mit bewegter Grafik oder bei der Steuerung einer Uhr bei Laborexperimenten), ist interpretiertes BASIC viel zu langsam.

Benötigen Sie in Ihren Programmen hohe Geschwindigkeiten, gibt es nur zwei Möglichkeiten: Entweder Sie programmieren in Maschinensprache oder Assembler-Sprache – ein schwieriger und zeitaufwendiger Vorgang – oder Sie verwenden eine Compiler-Sprache wie beispielsweise PASCAL oder FORTH. Compiler-Sprachen sind nicht sehr schwer zu erlernen, doch ist es so gut wie sicher, daß der Source Code Fehler enthält, die der Compiler während des Compilierungsvorganges findet. Im Gegensatz zu BASIC sind Fehler relativ schwer zu korrigieren. Nach der Berichtigung muß der Source Code vollkommen neu compiliert werden. Die meisten Compiler brauchen zwei oder drei Durchgänge (passes) durch den Source Code, wobei bei jedem Durchgang Fehler auftreten können.

Was kommt nach BASIC?

Ein korrekt compiliertes Programm zu erstellen ist ein außerordentlich zeitaufwendiger Prozeß. Andererseits verleitet BASIC dazu, sich schlechte Programmiertechniken anzugewöhnen, was bei höheren, strukturierten Sprachen wie PASCAL von Anfang an unmöglich ist. BASIC gestattet dem Programmierer nachlässig zu programmieren (z. B. durch die Verwendung vieler GOTOs). Diese schlechten Angewohnheiten können den Übergang zu fortgeschrittenen, strukturierten Sprachen schwierig machen.

BASIC ist eine flexible Sprache, die nicht schwer zu erlernen ist. Sie bietet ein sehr komfortables STRING-Handling, verhindert jedoch, die Fähigkeiten eines Heimcomputers voll auszunutzen. Modernere Programmiersprachen, wie PASCAL und FORTH, offerieren dagegen Programmiermöglichkeiten, die entweder zu schwer sind oder gar nicht in BASIC nachvollzogen werden können.

PASCAL wurde als Lehrsprache entwickelt und sollte speziell die Entwicklung gut konstruierter, strukturierter Programme unterstützen. PASCAL ist eine Compiler-Sprache, was für den Anwender bedeutet, daß er die durch den Compiler aufgedeckten zahlreichen Fehler jeweils einzeln beseitigen muß. Neulinge tendieren dazu, einige Punkte der Programmiersprache PASCAL als Einschränkung einer freien und flexiblen Programmierung zu empfinden, wie etwa die Notwendigkeit, alle Variablen am Anfang des Programmes festzulegen (um welchen Variablentyp es sich handelt – real, Integer usw.).

Außerdem verlangt PASCAL, daß sich der Programmierer bereits vor dem Schreiben des Ablaufs Gedanken über die Logik macht. Programme in PASCAL enthalten im Source Code fast zwangsläufig zahlreiche Syntax-Fehler. Dagegen sind logische und fundamentale Fehler relativ unwahrscheinlich.

FORTH ist als Programmiersprache für Heimcomputer eine populäre Alternative geworden. Obwohl FORTH eine höhere Programmiersprache ist, kommt ihre Ausführungsgeschwindigkeit aufgrund der einzigartigen Arbeitsweise der Maschinensprache sehr nahe. Wo bei anderen Programmiersprachen wie BASIC eine feste Anzahl an Anweisungen und Befehlen zur Verfügung steht, kann man bei FORTH, ähnlich wie bei LOGO, ein eigenes Vokabular definieren.

Vor- und Nachteile

Das Schlüsselwort PRINT in BASIC bedeutet, daß alle nachfolgenden Zeichen in Anführungszeichen auf dem Bildschirm ausgegeben werden. Daran kann der Programmierer nichts ändern. In FORTH dagegen läßt sich PRINT so definieren, daß beispielsweise eine Liste der hexadezimalen Äquivalente der ASCII-Codes eines Strings in einer vertikalen Spalte auf dem Bildschirm ausgegeben wird.

FORTH bietet dem Programmierer die Möglichkeit, jedes Wort wunschgemäß zu definieren und ihm jede beliebige Funktion zuzuordnen. FORTH ist in dieser Richtung nicht nur sehr flexibel, sondern produziert auch Programme, die im Object Code compiliert werden können. Diese sind dann nahezu so kompakt und schnell wie Maschinensprache-Programme. FORTH ist inzwischen für zahlreiche Heimcomputer erhältlich.

BASIC
Leicht zu erlernen
Leicht zu merken
Leichte Fehlerbeseitigung
Langsam in der Ausführung
Verbraucht viel Speicherplatz
Unterstützt keine strukturierte Programmierung

Assembler-Sprache
Nicht leicht erlernbar
Nicht leicht zu merken
Schwierige Fehlerbeseitigung
Sehr schnell in der Ausführung
Ermöglicht totale Kontrolle über die CPU

FORTH
Nicht sehr leicht erlernbar. Einfacher für totale Anfänger, schwieriger für BASIC-Programmierer
Relativ leicht zu merken
Fehlerbeseitigung im Interpreter-Modus sehr einfach
Kann compiliert werden. Geschwindigkeit fast so hoch wie bei ASSEMBLER-Sprache
Bietet komplette Kontrolle über die CPU
Sehr sparsam im Speicherplatzbedarf
Leichter zu erlernen als ASSEMBLER-Sprache, jedoch weniger „intuitiv" als BASIC

PASCAL
Relativ leicht erlernbar
Relativ leicht zu merken
Fehlerbeseitigung schwieriger als in BASIC
Unterstützt bessere Programmiertechniken
Ausführungsgeschwindigkeit schneller als BASIC, jedoch langsamer als ASSEMBLER
Muß compiliert werden, was Zeit kostet. Nach korrekter Compilierung ist die Geschwindigkeit fast so schnell wie bei ASSEMBLER.
Bietet eine gute Kontrolle über die CPU, jedoch nicht so gut wie ASSEMBLER.
Das String-Handling ist nicht so leicht wie bei BASIC.

Obwohl es noch viele weitere Programmiersprachen gibt, neigen die meisten Hobby-Programmierer dazu, nach BASIC entweder die ASSEMBLER-Sprache, PASCAL oder FORTH zu erlernen. Die Vor- und Nachteile der einzelnen Sprachen sind in Kurzform aufgeführt.

BASIC-Dialekte

LYNX 96

```
 1 REM *ERSTELLE DATEN-
   DATEI
 2 DIM N$ (30)
 3 LET N$="@ERST"
 4 DIM F$(15)
 5 LET F$="TEST"
 6 LET Z=2
 7 EXTBACK 1
 8 EXTSTORE 1,Z,N$,N$,N$
 9 EXTSTORE 1,F$,F$, F$, F$
10 INPUT "LEGE DATENCASSETTE
   EIN, DRUECKE RECORD UND
   'J'"; A$
11 SSAVE 1, "ADDBKDAT"
12 PRINT "STOPPE BAND, SPULE
   ZURÜCK"
13 END
```

ANMERKUNG: Dies ist das Initialisierungsprogramm für den 96K-Lynx.

Hauptprogramm-Variablen
Kopieren Sie die Spectrum-Liste mit den folgenden Änderungen für die numerischen Variablen:

Ersetze: GROSS durch Z
VMOD durch R
SRTD durch D
CURR durch C
WAHL durch H
BTM durch b
MD durch m
TP durch t

Außerdem müssen Sie die folgenden Ergänzungen und Änderungen durchführen:

```
1100 REM *CREARR* U/R
1110 DIM N$ (30) (50)
1120 DIM M$ (30) (50)
1130 DIM S$(30) (50)
1140 DIM T$(15) (50)
1150 DIM C$(15) (50)
1160 DIM R$(15) (50)
1170 DIM X$(15) (50)
1180 DIM Z$(30)
1210 LET Z=0
1220 LET R=0
1230 LET D=1
1240 LET C=0
1250 LET Z$="@ERST"
1260 LET Q$=""
1300 RETURN

1400 REM *LSINDT* U/R
1405 PRINT "DATENCASSETTE
     EINLEGEN UND PLAY
     DRUECKEN"
1410 GOSUB 3100
1420 SLOAD 1, "ADDBKDAT"
1430 PRINT "STOPPE BAND"
1440 GOSUB 3100
1450 EXTBACK 1
1460 EXTFETCH 1,Z
1470 FOR K=1 TO Z-1
1480 EXTFETCH 1, N$(K), M$(K),
     S$(K), T$(K), C$(K), R$(K),
     X$(K)
1490 NEXT K
1500 LET Q$=N$(1)
1510 RETURN

3120 IF KEYN <> 32 THEN LET L=0

3780 LET A$=KEY$

3810 LET H=VAL(A$)
3820 IF (H < 1) OR (H > 9) THEN
     LET L=0

4500 REM *MODNAM* U/R
4510 REM WANDLE IN GROSS-
     BUCHSTABEN UM
4520 LET D$=UPC$(N$(Z)) (lösche
     Zeilen 4530—4590)

4600 LET P$=""
4601 LET A$=""
4602 LET T=LEN(D$)
4603 LET S=0

4610 REM LOKALISIERE
     LETZTE LEERSTELLE

4630 IF MID$(D$,L,1)=" " THEN
     LET S=L

4670 IF MID$(D$,L,1)>"@" THEN
     LET P$=P$+MID$(D$,L, 1)

4710 IF MID$(D$, L, 1)> "@" THEN
     LET A$=A$+MID$(D$,L,1)
```

Die Zeilen 5410 bis 5460 müssen in einzelne Anweisungen umgewandelt werden. Z. B.:

```
5410 LET U$=N$(L):LET N$(L)=N$(T)
     :LET N$(T)=U$
```

wird zu

```
5410 LET U$=N$(L)
5411 LET N$(L)=N$(T)
5412 LET N$(T)=U$
```

Wandeln Sie in dieser Form alle Zeilen um.

```
5600 REM *SPEVER* U/R
5605 PRINT "LEGE DATENCASSETTE
     EIN UND DRUECKE RECORD"
```

```
5610 GOSUB 3100
5620 EXTBACK 1
5630 EXTSTORE 1,Z
5640 FOR K=1 TO Z-1
5650 EXTSTORE 1, N$(K), M$(K),
     S$(K), T$(K), C$(K), R$(K),
     X$(K)
5660 SSAVE 1, "ADDBKDAT"
5670 PRINT "STOPPE BAND"
5680 GOSUB 3100
5690 RETURN
.
5855 LET X=0
5860 LET m=INT((b+t)/2)
5870 IF M$(m)=U$ THEN LET X=1
5880 IF U$ > M$(m) THEN LET
     b=m+1
.
6080 LET A$=KEY$
.
6110 IF A$=" " THEN RETURN
6120 GOSUB 6200
6130 RETURN
.
6730 FOR I=1 TO 1
6735 LET I=0
6740 LET A$=KEY$
6750 IF (A$=E$) OR (A$=" ") THEN
     LET I=1
6760 NEXT I
```

Dieser Teil muß in den Zeilen 6880-6910, 6990-7030, 7110-7140, 7220-7250 und 7640-7670 reproduziert werden.

DRAGON 32

BBC MICRO

COMMODORE 64

VIC-20

Initialisierungsprogramm
Das Initialisierungsprogramm für den Dragon 32:

```
1 REM *ERSTELLE DATENDATEI
2 LET Z=2
3 LET N$="@ERST"
4 OPEN"0",#-1, "ADDBKDAT"
5 INPUT "DATENCASSETTE EIN-
  LEGEN, DRUECKE RECORD UND
  'J' ";A$
6 PRINT#-1,Z,N$,N$,N$,N$,N$,N$,N$
7 CLOSE #-1
8 PRINT "STOPPE BAND, SPULE
  ZURÜCK"
9 STOP
```

Beim Acorn B ersetzen Sie die Zeilen 4, 6 und 7 durch:

```
4 F1=OPENOUT("ADDBKDAT")
6 PRINT #F1,Z,N$,N$,N$,N$,N$,
  N$,N$
7 CLOSE#F1
```

Beim Commodore 64 und VC 20 ersetzen Sie die Zeilen 4, 6 und 7 durch:

```
4 OPEN  1,1,2,"ADBKDAT"
6 PRINT #1,Z:PRINT #1,N$:PRINT #1,
  N$:PRINT #1,N$:PRINT #1,
  N$:PRINT #1,N$:PRINT #1,
  N$:PRINT #1,N$
7 CLOSE 1
```

Hauptprogramm-Variablen
Beim Dragon, den Commodore-Computern und dem Acorn B kopieren Sie bitte die Spectrum-Liste und nehmen die folgenden Änderungen in Bezug auf die numerischen Variablen vor.

```
Ersetze: GROSS durch Z
         VMOD  durch R
         SRTD  durch D
         CURR  durch C
         WAHL  durch H
         BTM   durch BT
         MD    durch MD
         TP    durch TP
```

Außerdem sind die folgenden Änderungen und Ergänzungen auszuführen:

```
1100 REM *CREARR* U/R
1110 DIM N$(50)
1120 DIM M$(50)
1130 DIM S$(50)
1140 DIM T$(50)
1150 DIM C$(50)
1160 DIM R$(50)
1170 DIM X$(50)
```

Löschen Sie die Zeilen 1180 bis 1190.

```
1210 LET Z=0
1220 LET R=0
1230 LET D=1
1240 LET C=0
1250 LET Z$="@ERST"
1260 LET Q$=""
1300 RETURN
```

Dies ist die Dragon 32-Fassung der Unterroutine ab Zeile 1400:

```
1400 REM *LSINDT* U/R
1410 OPEN"I",#-1, "ADBKDAT"
1420 PRINT "DATENCASSETTE EIN-
     LEGEN UND PLAY DRUECKEN"
1430 GOSUB 3100
1440 INPUT#-1,Z
1450 FOR K=1 TO Z-1
1460 INPUT#-1,N$(K),M$(K),S$(K),
     T$(K),C$(K),R$(K),X$(K)
1470 NEXT K
1480 Q$=N$(1)
1490 CLOSE#-1
1500 PRINT "STOPPE BAND"
1510 GOSUB 3100
1520 RETURN
```

In der vorherigen Liste müssen Sie beim Acorn B Zeile 1410 wie folgt ersetzen:

```
1410 F1=OPENIN
     ("ADBKDAT")
```

Außerdem muß #-1 in den Zeilen 1440, 1460 und 1490 durch #F1 ersetzt werden.

Beim Commodore 64 und VC 20 muß Zeile 1410 wie folgt ersetzt werden:

```
1410 OPEN 1,1,0, "ADBKDAT"
```

Ersetzen Sie außerdem #-1 in den Zeilen 1440 und 1460 durch #1 sowie Zeile 1490 wie folgt:

```
1490 CLOSE 1
```

Beim Acorn B muß durchgehend INKEY$ durch INKEY$(0) ersetzt werden. Die Anweisungen INPUT "... Text . .";A$ müssen durch INPUT " . . .Text . .",A$ ersetzt werden. Bei den Commodore-Computern ersetzen Sie im gesamten Programm LET A$=INKEY$ durch GET A$. Außerdem muß IF INKEY$ . . . durch GET GT$:IF GT$ . . ersetzt werden.

Beim Acorn B, dem Dragon und den Commodore-Computern müssen in der Unterroutine ab Zeile 4500 alle Referenzen zu D$(L) durch MID$(D$,L,1) ersetzt werden. Ersetzen Sie CODE . . durch ASC(...). Ersetzen Sie in Zeile 4610 ERST durch LAST.
Löschen Sie: LET L=T ab Ende der Zeile 4630.

Dies ist die Dragon-Version der Unterroutine ab Zeile 5600. Für den Acorn B und die Commodore-Computer sehen Sie die Anmerkungen zur Initialisierung (s.o.).

```
5600 REM *SPEVER* U/R
5610 OPEN "O"#-1, "ADBKDAT"
5620 PRINT "DATENCASSETTE EIN-
     LEGEN UND RECORD
     DRUECKEN"
5630 GOSUB 3100
5640 PRINT#-1,Z
5650 FOR K=1 TO Z-1
5660 PRINT#-1,Z,N$(K),M$(K),S$(K),
     T$(K),C$(K),R$(K),X$(K)
5670 NEXT K
5680 CLOSE #-1
5690 PRINT "STOPPE BAND"
5693 GOSUB 3100
5695 RETURN
```

In der Unterroutine ab Zeile 6200 fügen Sie beim Acorn B ein:

```
6205 VDU 2
6275 VDU 3
```

Außerdem müssen Sie LPRINT durch PRINT ersetzen.

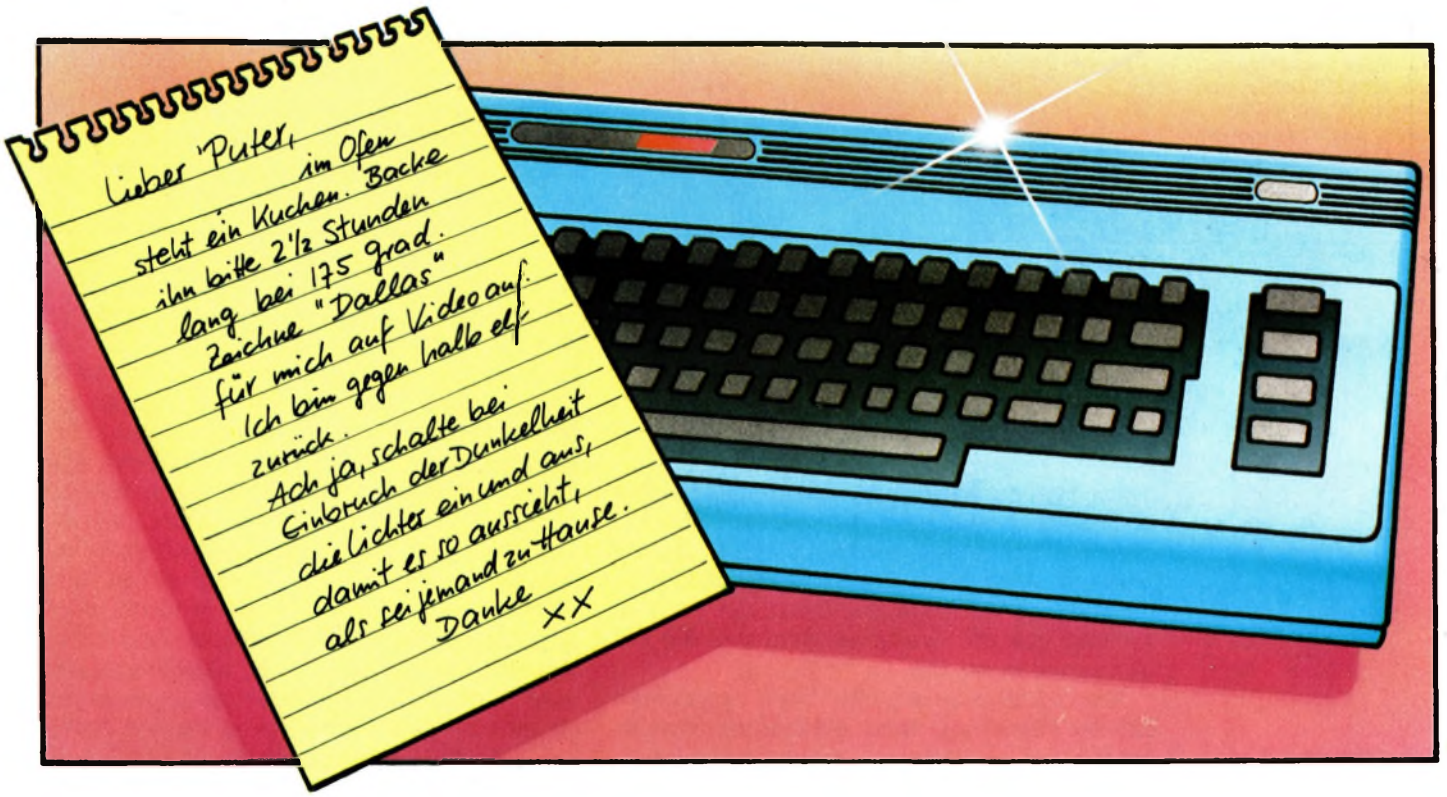
Bei den Commodore-Computern fügen Sie ein:

```
6205 OPEN 4,4:CMD 4
6275 PRINT #4:CLOSE 4
```

Außerdem müssen Sie LPRINT durch PRINT ersetzen.

Beim Dragon ersetzen Sie LPRINT durch PRINT #-2.

Steuern und Regeln



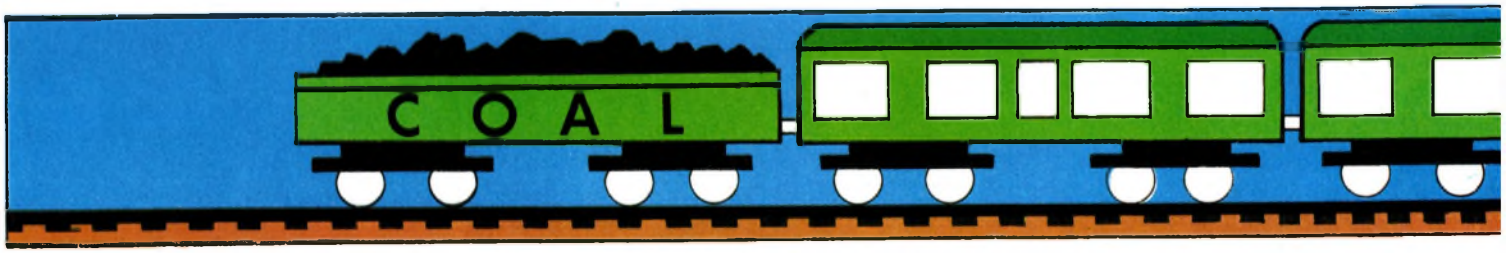
Heimcomputer-Besitzer können wählen, ob sie Software kaufen oder eigene Programme schreiben wollen. Doch nur wenige stellen Überlegungen an, ob sie Peripherien kaufen oder selbst bauen sollten. Und doch gibt es jetzt einige Komponenten auf dem Markt, die den Bau computergesteuerter Geräte ermöglichen.

Der einzige Grund, einen Microcomputer so zu programmieren, daß er am Morgen die Vorhänge öffnen oder während der Ferienzeit die Blumen gießen läßt, ist der Spaß an der Sache. Und das ist auch völlig in Ordnung. Warum sollte man nicht einige Stunden damit verbringen, eigene Software zu schreiben, wenn es Spaß macht?

Gegenwärtig mag der Bau eigener Peripherie-Geräte als „Herumspielen" mit selbstgemachten technischen Einrichtungen betrachtet werden. Doch langfristig gesehen wird sich dies als sinnvolle und nützliche Betätigung erweisen. Fähigkeiten, die man heute erlernt, werden in der Zukunft von unschätzbarem Wert sein. Und schließlich wurde so manche große Computer-Gesellschaft (wie Apple oder Atari) in einer Garage gegründet, von Leuten, die ganz einfach mit elektronischem „Kram" und Gerät „herumspielten".

In einem computergesteuerten System werden viele Elemente benötigt. Da ist zunächst der Computer selbst und das, was gesteuert werden soll. Außerdem sind bestimmte Dinge notwendig, um die Steuersignale zu übermitteln, und ferner Software, die den Computer in die Lage versetzt zu entscheiden, was diese Meldungen oder Signale beinhalten sollen. Doch das ist nur ein Teil des Ganzen. Der Computer muß mit einer Vorrichtung ausgestattet

Durch das Verbinden eines Computers mit entsprechenden Geräten gibt es eine Fülle von Möglichkeiten für automatische, programmgesteuerte Erledigung verschiedener Aufgaben. Der Computer kann zu vorprogrammierten Zeiten oder bei Temperatur- oder Lichtveränderungen reagieren.



Schnittstellen für Steuerzwecke gibt es für viele Heimcomputer. Dabei handelt es sich zumeist um Schalteinheiten auf Relaisbasis. Der Computer kann die Stromzufuhr zu Geräten ein- oder ausschalten und erhält Signale, abhängig davon, ob der Sensor „an" oder „aus" ist.

sein, die ihm zu berechnen ermöglicht, was sein Steuerbefehl bewirkt hat, und ihm zugleich Korrekturmöglichkeiten gibt. Diesen Vorgang nennt man „Feedback" (Rückkoppelung). Ohne ihn wäre der Computer etwa so befähigt wie ein Autofahrer, dem man die Augen zugebunden hat.

Alle computergesteuerten Systeme basieren auf der Steuerung durch elektrische Signale. Leider sind die computerinternen elektrischen Signale zu schwach, um etwas bewirken zu können. Selbst eine winzige Glühbirne benötigt mehr Strom als irgendein Teil des Computers. Also ist eine grundlegende Voraussetzung, die minimalen Spannungen des Computers in größere zu verwandeln.

Der erste Schritt erfolgt im Computer selbst. Er braucht ein Gerät, mit dem Signale nach außen geschickt werden können. Dies geschieht, indem man einen bestimmten Teil des Computerspeichers ausschließlich für diesen Zweck benutzt. Über den „User Port" lassen sich bestimmte Informationen von außen lesen, ohne die interne Verarbeitung zu stören.

Einige Microcomputer sind standardmäßig mit einem solchen Interface ausgestattet, andere können nachträglich damit ausgerüstet werden. Der User Port selbst kann zum Ein- und Ausschalten von LEDs Anwendung finden. Doch für praxisnahe Systeme sind andere Komponenten erforderlich. Am sinnvollsten ist der Anschluß einiger elektronischer Teile sowie einer zusätzlichen Stromquelle, die die Steuerung von Relais erlaubt. Relais sind Schalter, die relativ große elektrische Ströme an- und abschalten können, selbst aber durch sehr kleine Ströme gesteuert werden.

Die meisten von Bastlern benutzten Relais arbeiten nur mit batteriebetriebenen Geräten. Es gibt wohl nur wenige Leute, die tatsächlich einen Bedarf an Relais für große Stromstärken haben. Da zudem höhere Spannungen sehr gefährlich sind, sollten ausschließlich auf technische Sicherheit überprüfte (VDE-getestete) Komponenten verwendet werden. Wer mit kleinen Spannungen arbeitet, kann zwischen fertig konfektionierten und selbst gebauten Relais-Schaltern, die an den Computer angeschlossen werden können, entscheiden.

Zur Zeit muß der Computer noch direkt mit den zu steuernden Geräten verbunden werden, was zweifelsfrei eine Einschränkung ist. Verschiedene Unternehmen arbeiten an der Lösung dieses Problems. Man entwickelt zum Beispiel Kabel, die nicht nur Strom leiten, sondern gleichzeitig Daten übertragen können. Mit diesem System ist es möglich, daß der Computer in einem zentralen Raum steht und Signale an sogenannte „Slaves" schickt, die ganz normal an beliebige Steckdosen angeschlossen werden. Der Computer sendet individuelle Anweisungen an die einzelnen Slaves (oder Verbraucherterminals), die ihnen sagen, ob sie ein- oder ausgeschaltet werden sollen. Jedes normale Haushaltsgerät, ob Stehlampe, Fernseher oder Heizofen, kann einfach an den Slave angeschlossen und so durch den Computer gesteuert werden.

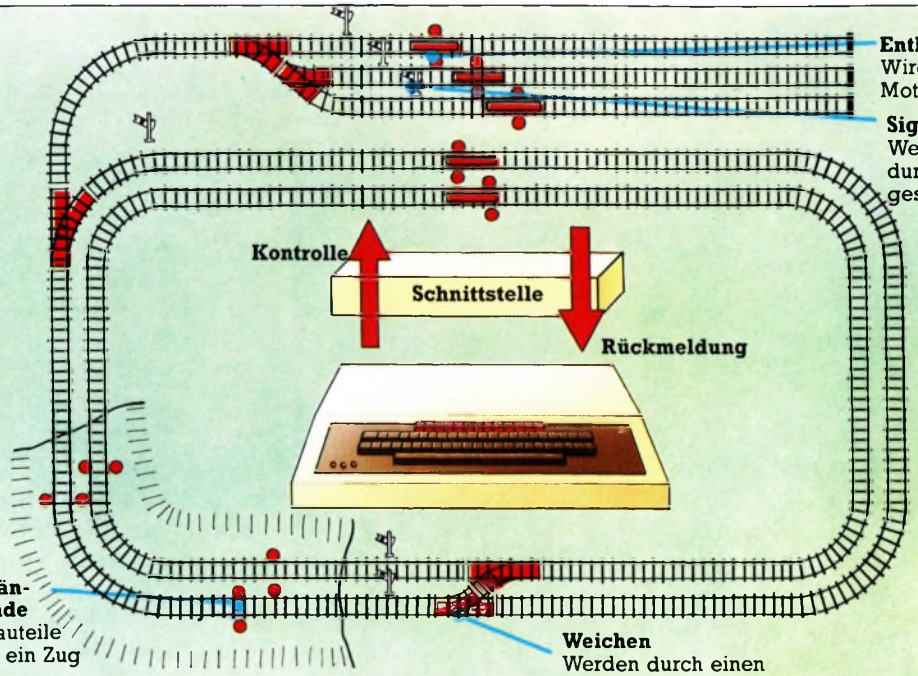
Rückkopplung ist das A und O

Es wurde schon darauf hingewiesen, daß fast alle computergesteuerten Systeme eine Rückmeldung benötigen, um die Funktionsweise des Systems überprüfen zu können. Auf den ersten Blick scheint nicht einsehbar, warum der Computer eine Rückmeldung dafür braucht, ob ein Licht ein- oder ausgeschaltet ist – sinnvoller wäre es, wenn der Computer wüßte, wann es draußen dunkel ist, und aufgrund dieser Information das Licht einschaltet.

Es gibt zwei Arten von Rückmeldungssignalen. Das kann einmal der Zustand „Ein" oder „Aus" sein, ohne daß Zwischenwerte erfaßt werden. Ein solches Signal kann ein einfacher Schalter sein, der meldet, ob ein Fenster geöffnet oder geschlossen ist oder ob die Türklingel betätigt wurde. User Ports können diese Ein/Aus-Signale lesen.

Eisenbahn-Steuerung

Bei jeder Art von Steuerung durch einen Computer – gleich ob Modelleisenbahn oder die elektrischen Anlagen eines ganzen Hauses – findet dieselbe Technik Anwendung. Man installiert eine Schleife, mit der der Computer über eine Schnittstelle Signale aussendet. Diese steuern Servomotoren, Lichter und so weiter. Das Gerät gibt dann eine Rückmeldung. Diese Rückmeldungs-Schleife ermöglicht eine exakte Steuerung durch den Computer.



Entkupplungsanlage
Wird durch einen Motor gesteuert.

Signale
Werden ebenfalls durch kleine Motoren gesteuert.

LED / lichtabhängige Widerstände
Diese beiden Bauteile melden, ob hier ein Zug vorhanden ist.

Weichen
Werden durch einen kleinen Motor gesteuert.

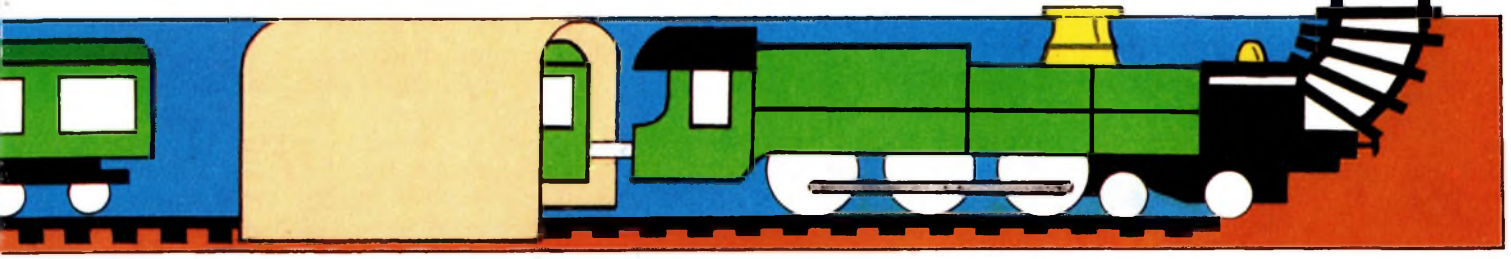
Nützlicher, aber auch komplizierter ist die als Analog-Signal bezeichnete Art von Rückmeldung. Ein solches Signal kann aus einer Reihe von Werten bestehen und so zur Temperaturmessung, zum Messen des Abstands eines Objekts oder wie weit es sich bewegt oder gedreht hat, verwendet werden. Aber auch die Gewichtsmessung oder die Messung der Spannungsabgabe einer Batterie ist möglich. Diese Art von Signal wird von einem Analog-Digital-Konverter (Umwandler) erkannt.

Allein von den Rückmeldungseinheiten hängt ab, was computergesteuert möglich ist. Ein Motor, der ein Rad dreht, kann von einem Computer überwacht werden, indem dieser feststellt, wie weit sich das Rad in einer bestimmten Zeit gedreht hat. Das funktioniert aber nicht, wenn eine Last auf das Rad gelegt wird oder die den Motor antreibenden Batterien schwächer werden, da das Rad sich dann langsamer dreht. Ein optischer Sensor könnte dem Computer in diesem Fall ein Signal übermitteln, wenn eine Umdrehung vollendet ist.

Einige Arten von Elektromotoren sind mit einer Rückmeldungseinheit ausgestattet. Das bedeutet: Der Computer sendet ein Signal, mit dem die Anweisung zur Bewegung in eine bestimmte Position gegeben wird, und der Motor arbeitet so lange, bis er diese erreicht hat. Es gibt zwei Arten dieser Motoren: Schritt- und Servomotoren. Ein Schrittmotor kann sich, wie jeder andere Motor, ständig drehen oder aber in jeder beliebigen Position angehalten werden. Er ist jedoch nicht sehr leistungsfähig und kann deshalb nur gering belastet werden. Servomotoren sind stark, können sich aber nur in einem kleinen Winkel bewegen, der normalerweise bei etwa 90 Grad liegt. Sowohl Schritt- als auch Servomotoren benötigen eine spezielle Steuereinheit, um über Computer angesprochen werden zu können.

Die letzte Gattung computergesteuerter Geräte, die hier vorgestellt werden soll, benötigt zur Steuerung wechselnde Spannungen. Ein Beispiel dafür wäre ein kleiner Elektromotor, der sich mit unterschiedlicher Geschwindigkeit abhängig von der zugeführten Stromspannung dreht. Das Gegenstück eines A/D-Wandlers, ein Digital-Analog-Wandler (kurz D/A), wandelt die digitalen Signale des Computers in wechselnde Stromspannungen um.

Der Einsatz von Computern zur Steuerung anderer Geräte ist dem Schreiben eigener Software vergleichbar. Man muß eine gute Idee mit technischem Wissen verbinden und umsetzen können, und man braucht Zeit. Die Ergebnisse mögen zwar nicht dem aktuellen technischen Stand entsprechen, doch es macht auf jeden Fall mehr Spaß, etwas „selbst zu tun", als fertige Massenprodukte zu kaufen.



Trigonometrie

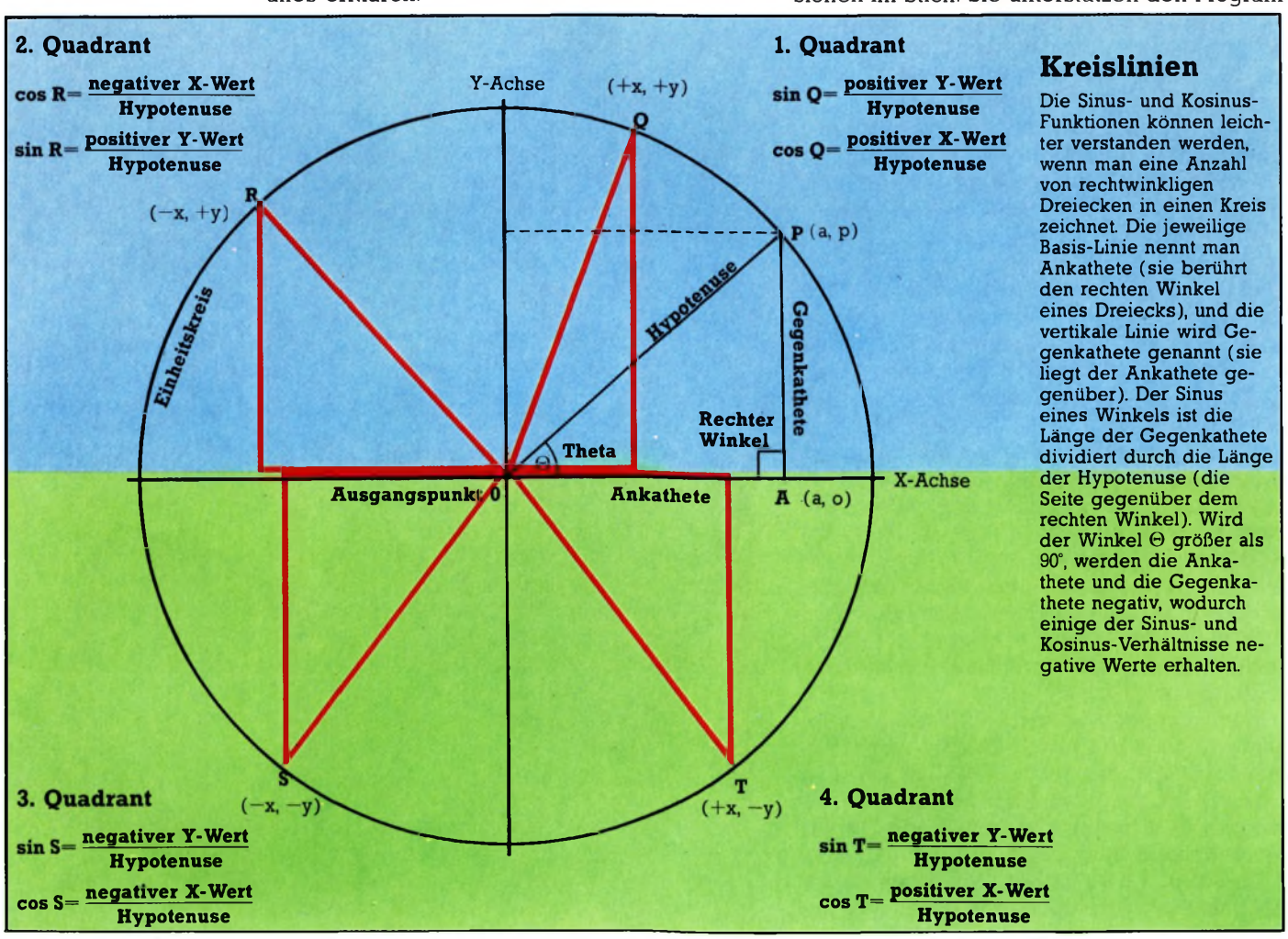
Oft ist es nicht zu vermeiden, mathematische Ausdrücke in den Programmen zu verwenden. Hier werden die Winkelfunktionen Sinus und Kosinus erklärt.

Welche Art der Mathematik müssen Programmierer kennen? Das hängt von der Art des Programms ab, das sie schreiben wollen. Die BASIC-Versionen, die in den meisten Heimcomputern integriert sind, umfassen viele Anweisungen und Funktionen für den Einsatz von Bildschirm-Grafiken – PLOT, CIRCLE, FILL, LINE, COLOUR, INK, PAPER usw. –, so daß beim Verschieben und Bewegen einfacher Figuren auf dem Bildschirm keine Probleme entstehen. Das ist meist sogar dann einfach, wenn trigonometrische Funktionen wie COS, SIN und TAN gebraucht werden, vorausgesetzt, Ihre BASIC-Version ist mit diesen Funktionen ausgerüstet. Wenn Sie die hier verwendeten Ausdrücke zunächst nicht verstehen sollten, haben Sie keine Angst – wir werden alles erklären.

Wie sieht es bei BASIC aus, wenn man statistische Funktionen braucht? Die Antwort lautet: sehr schlecht! Die meisten Versionen dieser Sprache verfügen nicht über eingebaute statistische Funktionen, die bei der Manipulation von Daten helfen könnten. Wenn Ihr Programm vorhersagen soll, wer die nächste Wahl gewinnt oder ob blauäugige Kinder im Examen besser abschneiden, müssen Sie die entsprechenden Funktionen selbst programmieren.

Unterstützung für BASIC

Auch wenn Sie Spiele oder Schreibmaschinenkurse schreiben, wobei es darauf ankommt, die Antwortzeit des Benutzers genau zu erfassen, lassen Sie die meisten BASIC-Versionen im Stich. Sie unterstützen den Program-



Kreislinien

Die Sinus- und Kosinus-Funktionen können leichter verstanden werden, wenn man eine Anzahl von rechtwinkligen Dreiecken in einen Kreis zeichnet. Die jeweilige Basis-Linie nennt man Ankathete (sie berührt den rechten Winkel eines Dreiecks), und die vertikale Linie wird Gegenkathete genannt (sie liegt der Ankathete gegenüber). Der Sinus eines Winkels ist die Länge der Gegenkathete dividiert durch die Länge der Hypotenuse (die Seite gegenüber dem rechten Winkel). Wird der Winkel Θ größer als 90°, werden die Ankathete und die Gegenkathete negativ, wodurch einige der Sinus- und Kosinus-Verhältnisse negative Werte erhalten.

mierer nicht einmal mit den einfachsten Zeitfunktionen. Und das sind dann auch die drei Hauptgebiete, mit denen wir uns in den folgenden Artikeln beschäftigen wollen – Trigonometrie, Statistik und Zeitmessung.

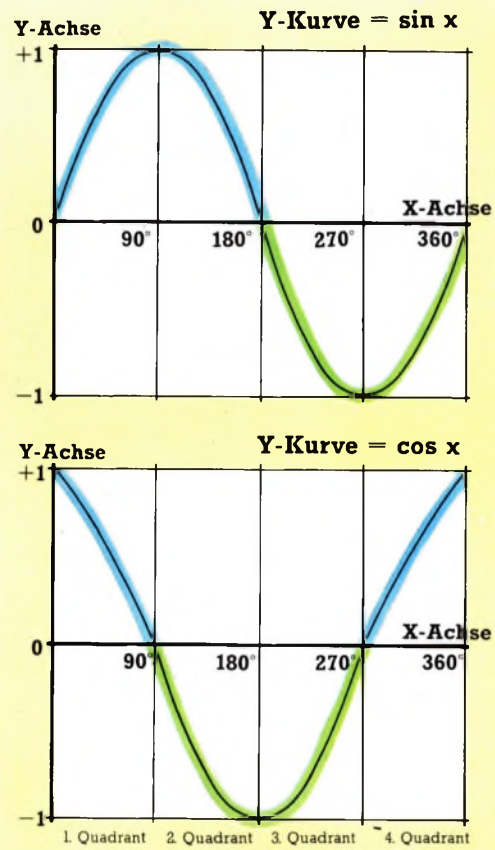
Mathematikschüler fragen sich oft, welche Bedeutung die Trigonometrie für die wirkliche Welt hat. Die Trigonometrie stellt die Verbindung zwischen der euklidschen Geometrie, die sich mit der Manipulation von Punkten, Linien und Kurven beschäftigt, und der Algebra, die mathematische Lösungen durch die Manipulation von Variablen mit unbekannten Werten ermöglicht, dar. Nehmen Sie als Beispiel eine Parabel. Ihre besonderen Eigenschaften lassen sich mit Hilfe von Millimeterpapier, Winkelmesser, Lineal und Bleistift herausfinden. Trotzdem ist es natürlich einfacher, die Kurve durch die algebraische Formel $y=x^2$ auszudrücken.

Diese einfache Formel gestattet, Werte für jeden beliebigen Punkt auf der Kurve zu berechnen, ohne daß man sie zeichnen müßte. Probleme, die algebraisch gelöst werden können, sind auch auf einem Computer erheblich einfacher zu lösen als solche, die gezeichnete Grafiken oder Figuren erfordern.

Kosinus und Sinus sind zwei Wege, das Verhältnis der Seiten eines rechtwinkligen Dreiekkes zueinander darzustellen. Jedes rechtwinklige Dreieck kann so gezeichnet werden, daß es genau in einen Kreis, genannt der Einheitskreis, hineinpaßt. Die Bezeichnung Einheitskreis resultiert daraus, daß er einen Radius von „1 Einheit" hat – das wirkliche Maß spielt dabei keine Rolle. Unser Bild zeigt eine Linie, die um 35° gedreht wurde. Der Startpunkt für eine Drehung ist vereinbarungsgemäß die horizontale Achse, und die Drehung erfolgt gegen den Uhrzeigersinn. Die horizontale Achse nennt man die X-Achse. Der Winkel einer Drehung heißt Theta (Θ). Wenn eine Linie vom Kreisumfang zur X-Achse gezogen wird, erhalten Sie ein rechtwinkliges Dreieck.

Gleichbleibende Verhältnisse

Der Kosinus von Θ ist definiert als das Verhältnis der Länge der am Winkel anliegenden Seite eines Dreiecks (der Teil parallel zur X-Achse, genannt die Ankathete) zur Hypotenuse (der Radius des Einheitskreises). Wenn wir beispielsweise einen Kreis mit einem Radius von 15 mm nehmen und die am Winkel anliegende Seite des Dreiecks messen, erhalten wir als Ergebnis annähernd 12,28728 mm. Dividiert man diesen Wert durch 15, so erhält man als Ergebnis den Wert 0,819152 und somit den Kosinus von 35°. Dieses Verhältnis bleibt bestehen, ganz gleich, welche Größe der Einheitskreis und das in ihn gezeichnete Dreieck haben mögen. Ob der Radius des Kreises nun einen Zentimeter, einen Kilometer oder ein Lichtjahr beträgt, die Seite des Dreiecks, die an der X-Achse liegt, wird immer ungefähr 0,82 des Radius-Wertes betragen.



Eine Kurve

Die bekannte Sinus-Kurve wird durch die grafische Darstellung der Sinus-Funktion eines kompletten Kreises erzeugt. Entlang der X-Achse reichen die Werte der Winkel von 0° bis 360°, und die Y-Achse repräsentiert den Bereich der Sinus-Werte für diese Winkel. Beachten Sie, daß alle Sinus-Werte in einem Bereich von plus 1 bis minus 1 liegen. Die vier Abschnitte des Diagramms entsprechen den vier Quadranten des in der anderen Abbildung gezeigten Kreises. Der Sinus ist positiv (blaue Linie) für die ersten beiden Quadranten und negativ (grüne Linie) für die anderen Quadranten. Die Kosinus-Kurve hat dieselbe Form wie die Sinus-Kurve, ist jedoch auf der X-Achse um 90° verschoben.

Wir könnten ebenso andere Werte von Θ in den hier gezeigten Einheitskreis einzeichnen. Sie werden feststellen, daß für jeden Wert von Θ der Wert von cos Θ niemals größer als 1 oder kleiner als 0 sein wird. Für Werte von Θ größer als 90° wird cos Θ allerdings einen negativen Wert annehmen. Dies kommt daher, daß cos Θ den Wert der X-Koordinate beeinflußt. Mathematische Definitionen legen fest, daß der Ursprungspunkt einer Drehung bei 0 auf der X-Achse liegen muß. Alle Punkte links dieser Achse haben somit negative Werte. Aus demselben Grund ist der Kosinus von Winkeln zwischen 180° und 270° ebenfalls negativ, wogegen der Kosinus von Winkeln größer als 270° bis zu 360° wieder positive Werte hat.

Die Sinus-Funktion eines Winkels ist der des Kosinus sehr ähnlich, allerdings werden dabei Werte auf der Y-Achse beeinflußt. Wenn Θ den Wert 0 hat, so ist die anliegende Seite des Dreiecks gleich der Länge der Hypotenuse. Die Koordinate von P auf der X-Achse wird immer 1 (da $1/1 = 1$), die Koordinate auf der Y-Achse dagegen 0 sein. Für alle Werte von Θ bis 90° liegt der sin Θ in einem Bereich von 0 bis 1. Im zweiten Quadranten des Kreises ist sin Θ ebenfalls positiv, nimmt jedoch abfallende Werte von 1 bis 0 an, entsprechend der Annäherung von Θ zu 180°. Alle Werte von Θ größer als 180° bis zu (nicht einschließlich) 360° sind dann negativ.

Grade der Präzision

Im zweiten Teil unserer Serie über Mathematik bei der BASIC-Programmierung fahren wir mit der Betrachtung von trigonometrischen Funktionen fort. Wie können die Sinus- und Kosinus-Funktionen in BASIC-Programmen verwendet werden?

Da BASIC sowohl mit der Sinus- als auch mit der Kosinus-Funktion ausgestattet ist, sollte die Berechnung der Position eines Punktes auf einer Linie nach einer Drehung um einen bestimmten Winkel eigentlich kein Problem darstellen. Der cos von Θ gibt die Position auf der X-Achse (X-Koordinate) an, und der sin von Θ die Position auf der Y-Achse (Y-Koordinate). Bei der Verwendung der beiden Funktionen müssen Sie daran denken, daß die meisten BASIC-Versionen mit dem Bogenmaß (Radiant) und nicht mit Grad arbeiten.

Zunächst soll versucht werden, den Unterschied zwischen Grad und Radiant zu überbrücken. Wenn ein Teil eines Kreises (genannt ein Bogen) so gezeichnet wird, daß seine Länge exakt dem Radius des Kreises entspricht, ist der Winkel im Zentrum ein Radiant (siehe Bild). Wenn der Radius des Kreises eine Einheit ist, hat dieser Teil des Kreisumfanges ebenfalls die Länge von einer Einheit. Die Formel zur Ermittlung des Kreisumfanges lautet $2\pi r$. Das bedeutet, daß eine komplette Umdrehung 2π Radianten sein müssen. Wie Sie sehen, entsprechen also 360° genau 2π Radianten. Dies gestattet uns nun, Grad und Radiant zueinander ins Verhältnis zu setzen:

360° = 2π Radianten
180° = π Radianten
90° = $\pi/2$ Radianten
1° = $\pi/180$ = 0,0174 Radianten

Ein BASIC-Programm, das den Kosinus eines in Grad gemessenen Winkels herausfinden soll, muß zuerst das Winkelmaß von Grad in Radiant umrechnen und dann die cos-Funktion anwenden. Testen Sie das folgende Programm:

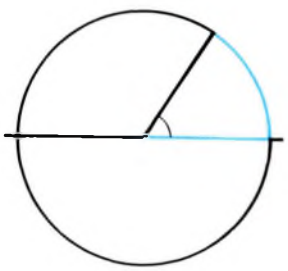
```
10 INPUT "GIB DEN WINKEL IN GRAD
   AN";A
20 LET B# = A * 0.0174
30 LET C# = COS(B#)
40 PRINT "DER KOSINUS VON ";A;" GRAD
   BETRAEGT ";C#
50 END
```

Das Nummernzeichen bedeutet, daß die Variablen in diesem Programm doppelt genau sind. Eine einfache Änderung an diesem Programm unter Verwendung der sin-Funktion liest alle Werte von theta von 0° bis 360° ein und gibt die jeweiligen Sinus-Werte in Form einer Tabelle aus. Werden diese Werte entlang der Y-Achse eines Diagramms dargestellt, erhält man die Sinus-Kurve. Diese Kurve ist nichts anderes als die Darstellung aller Positionen der Schnittstelle der Hypotenuse mit dem Einheitskreis für alle Winkel einer Drehung. Mit anderen Worten: Dies ist ein alternativer Weg, einen Kreis mathematisch zu beschreiben.

Einige BASIC-Versionen erlauben, daß die sin- und cos-Funktionen sowohl mit Grad als auch mit Radiant arbeiten, indem man eine spezielle Funktion aufruft. Bei den meisten Versionen ist dies jedoch nicht möglich. Wenn Sie es vorziehen, ständig mit Grad zu arbeiten, können Sie eine Funktion definieren, die die Umrechnung für Sie erledigt. Das folgende Programm stellt eine Möglichkeit dar:

```
10 REM FUNKTION FUER
   BERECHNUNGEN IN GRAD
20 DEF FNGSIN (D#)=SIN
   (D#*0.017453293)
30 INPUT "GIB DEN WINKEL IN GRAD
   AN";D#
40 PRINT "DER SINUS VON #";D#;" GRAD
   BETRAEGT ";FNGSIN(D#)
50 END
```

Ein Radiant



Fehler bei der Sinus-Funktion

```
 1 REM TEST FUER DIE BERECHNUNG DER SIN-FUNKTION
10 LET X=10/6
20 PRINT "WIEDERHOLUNG", "WERT VON X", "WERT VON SIN(X)"
30 FOR I=1 TO 40
40 LET X=X/10
50 PRINT I,X,SIN(X)
60 NEXT I
70 END
```

WIEDERHOLUNG	WERT VON X	WERT VON SIN(X)
1	.166667	.165896
2	.0166667	.0166659
3	1.66667E-03	1.66667E-03
4	1.66667E-04	1.66667E-04
5	1.66667E-05	1.66667E-05
6	1.66667E-06	1.66667E-06
7	1.66667E-07	1.66667E-07
8	1.66667E-08	1.66667E-08
9	1.66667E-09	1.66667E-09
10	1.66667E-10	1.66667E-10
11	1.66667E-11	1.66667E-11
12	1.66667E-12	1.66667E-12
13	1.66667E-13	1.66667E-13
14	1.66667E-14	1.66667E-14
15	1.66667E-15	1.66667E-15
16	1.66667E-16	1.66667E-16
17	1.66667E-17	1.66667E-17
18	1.66667E-18	1.66667E-18
19	1.66667E-19	1.66667E-19
20	1.66667E-20	1.66667E-20
21	1.66667E-21	1.66667E-21
22	1.66667E-22	1.66667E-22
23	1.66667E-23	1.66667E-23
24	1.66667E-24	1.66667E-24
25	1.66667E-25	1.66667E-25
26	1.66667E-26	1.66667E-26
27	1.66667E-27	1.66667E-27
28	1.66667E-28	1.66667E-28
29	1.66667E-29	1.66667E-29
30	1.66667E-30	1.66667E-30
31	1.66667E-31	1.66667E-31
32	1.66667E-32	1.66667E-32
33	1.66667E-33	1.66667E-33
34	1.66667E-34	1.66667E-34
35	1.66667E-35	1.66667E-35
36	1.66667E-36	1.66667E-36
37	1.66667E-37	1.66667E-37
38	1.66667E-38	1.66667E-38
39	0	0
40	0	0

In Zeile 20 wird eine Funktion GSIN aufgerufen (GSIN steht für GRAD/SINUS), die als Parameter die doppelt genaue Variable D# verwendet. Die rechte Hälfte der Definition zeigt lediglich, wie der Wert, der durch die Funktion zurückgegeben werden soll (der Sinus eines Winkels in Grad), abgeleitet wird.

Eines der Probleme bei Verwendung der Sinus-Funktion in BASIC ist, daß nicht alle BASIC-Versionen sie korrekt handhaben, wenn der Wert von theta sich 0 nähert. BASIC arbeitet nur in einem bestimmten Zahlenbereich sehr genau. Wenn theta sehr klein ist (zum Beispiel 1,0E-36 = 1 × 10 hoch Minus 36), ist BASIC überfordert und gibt einfach einen Wert 0 als Ergebnis für den Sinus einer solchen Zahl aus. Bevor Sie die Sinus-Funktion ausprobieren, testen Sie zuerst einmal Ihr BASIC mit dem Programm der vorherigen Seite.

Wir haben einen Beispiellauf dieses Programmes in Microsoft-MBASIC beigefügt. Dieser spezielle BASIC-Interpreter bearbeitet den sin von kleinen Zahlen recht genau und verursacht keine Probleme, bis zu einem Wert von Θ kleiner als 1,0E-38 (ein Dezimalpunkt gefolgt von 37 Nullen).

Das gezeigte Programm ist abhängig von einem angemessenen dynamischen BASIC-Bereich in bezug auf arithmetische Fließkomma-Operationen. Es ist also ratsam, vor Verwendung mathematischer Operationen zuerst einmal den Bereich der Zahlen, der korrekt verarbeitet werden kann, zu überprüfen.

Beachten Sie auch, daß ein Variablenname allein, wie beispielsweise X oder TREND, automatisch nur einfach genau ist (das heißt nicht mehr als sieben Stellen speichern kann). Variablen können auch von vornherein als einfach genau spezifiziert oder dazu „gemacht" werden, indem man ein Ausrufungszeichen anhängt (in diesem Fall X! oder TREND!). Doppelt genaue Variablen können 17 Stellen speichern. Sie werden spezifiziert, indem man ein Nummernzeichen anhängt, wie bei X# oder TREND#. Integer-Variablen werden bei vielen BASIC-Versionen spezifiziert, indem dem Variablennamen ein Prozentzeichen angehängt wird, hier beispielsweise X% oder TREND%.

Wir beenden diesen Artikel mit einem kurzen Programm, mit dem Sie testen können, wie viele Stellen in einer Variable Ihrer BASIC-Version gespeichert werden können. Zusätzlich zeigen wir Ihnen einen Ausdruck eines Programmlaufs unter Verwendung von Microsoft-BASIC. Es gibt zwei Versionen des Programms, eine zum Testen von kleinen Zahlen und eine für große Zahlen. Der Ausdruck für die kleinen Zahlen zeigt, daß BASIC die Zahlen zu 0 rundet, wenn die Zahlen sehr klein werden (kleiner als 3,3×10E-38). Für große Zahlen (größer als 3,3×10E37) tritt ein Überlauf ein, und die Ergebnisse sind nicht mehr exakt. Wenn Sie mit sehr großen oder sehr kleinen Zahlen arbeiten müssen, kann es sein, daß Sie spezielle Routinen entwickeln müssen, um diese Einschränkungen zu umgehen.

Kleine Zahlen in BASIC

```
1 REM TESTET DIE HANDHABUNG VON KLEINEN ZAHLEN IN BASIC
10 LET X# = .000033333333333#
20 PRINT "WIEDERHOLUNG","  DOPPELT GENAU","  "," EINFACH GENAU"
30 PRINT
40 FOR I=1 TO 40
50 LET X#=X#/10
60 LET X!=X#
70 PRINT I,X#,X!
80 NEXT I
90 END
```

WIEDERHOLUNG	DOPPELT GENAU	EINFACH GENAU
1	.000003333333333	3.33333E-06
2	.0000003333333333	3.33333E-07
3	3.333333333D-08	3.33333E-08
4	3.333333333D-09	3.33333E-09
5	3.333333333D-10	3.33333E-10
6	3.333333333D-11	3.33333E-11
7	3.333333333D-12	3.33333E-12
8	3.333333333D-13	3.33333E-13
9	3.333333333D-14	3.33333E-14
10	3.333333333D-15	3.33333E-15
11	3.333333333D-16	3.33333E-16
12	3.333333333D-17	3.33333E-17
13	3.333333333D-18	3.33333E-18
14	3.333333333D-19	3.33333E-19
15	3.333333333D-20	3.33333E-20
16	3.333333333D-21	3.33333E-21
17	3.333333333D-22	3.33333E-22
18	3.333333333D-23	3.33333E-23
19	3.333333333D-24	3.33333E-24
20	3.333333333D-25	3.33333E-25
21	3.333333333D-26	3.33333E-26
22	3.333333333D-27	3.33333E-27
23	3.333333333D-28	3.33333E-28
24	3.333333333D-29	3.33333E-29
25	3.333333333D-30	3.33333E-30
26	3.333333333D-31	3.33333E-31
27	3.333333333D-32	3.33333E-32
28	3.333333333D-33	3.33333E-33
29	3.333333333D-34	3.33333E-34
30	3.333333333D-35	3.33333E-35
31	3.333333333D-36	3.33333E-36
32	3.333333333D-37	3.33333E-37
33	3.333333333D-38	3.33334E-38
34	0	0
35	0	0
36	0	0
37	0	0
38	0	0
39	0	0
40	0	0

Große Zahlen in BASIC

```
1 REM TESTET DIE HANDHABUNG VON GROSSEN ZAHLEN IN BASIC
10 LET X# = 3.333333333333334#
20 PRINT "WIEDERHOLUNG","  DOPPELT GENAU","  " EINFACH GENAU"
30 PRINT
40 FOR I=1 TO 40
50 LET X#=X#*10
60 LET X!=X#
70 PRINT I, X#,X!
80 NEXT I
90 END
```

WIEDERHOLUNG	DOPPELT GENAU	EINFACH GENAU
1	33.33333333333334	33.3333
2	333.3333333333334	333.333
3	3333.333333333334	3333.33
4	33333.33333333334	33333.3
5	333333.3333333334	333333
6	3333333.333333334	3.33333E+06
7	33333333.33333334	3.33333E+07
8	333333333.3333334	3.33333E+08
9	3333333333.333334	3.33333E+09
10	33333333333.33334	3.33333E+10
11	333333333333.3334	3.33333E+11
12	3333333333333.334	3.33333E+12
13	33333333333333.34	3.33333E+13
14	333333333333333.4	3.33333E+14
15	3333333333333334	3.33333E+15
16	3.333333333333334D+16	3.33333E+16
17	3.333333333333334D+17	3.33333E+17
18	3.333333333333334D+18	3.33333E+18
19	3.333333333333334D+19	3.33333E+19
20	3.333333333333334D+20	3.33333E+20
21	3.333333333333334D+21	3.33333E+21
22	3.333333333333334D+22	3.33333E+22
23	3.333333333333334D+23	3.33333E+23
24	3.333333333333334D+24	3.33333E+24
25	3.333333333333334D+25	3.33333E+25
26	3.333333333333334D+26	3.33333E+26
27	3.333333333333334D+27	3.33333E+27
28	3.333333333333334D+28	3.33333E+28
29	3.333333333333334D+29	3.33333E+29
30	3.333333333333334D+30	3.33333E+30
31	3.333333333333335D+31	3.33333E+31
32	3.333333333333335D+32	3.33333E+32
33	3.333333333333334D+33	3.33333E+33
34	3.333333333333335D+34	3.33333E+34
35	3.333333333333335D+35	3.33333E+35
36	3.333333333333335D+36	3.33333E+36
37	3.333333333333335D+37	3.33333E+37
38	1.701411834604693D+38	1.70141E+38
39	1.701411834604693D+38	1.70141E+38
40	1.701411834604693D+38	1.70141E+38

Mensch und Maschine

Die Ergonomie befaßt sich mit der Anpassung von Handwerkszeug und Umfeld an die Bedürfnisse des arbeitenden Menschen, unter Berücksichtigung unterschiedlichster Kriterien. Dabei sind physikalisch-technische Aspekte ebenso wie die der Physiologie und Psychologie zu beachten.

Bei der „Schnittstelle Benutzer" denkt jeder zunächst an die Tastatur, die Gegenstand vieler ergonomischer Verbesserungsversuche war und ist – Konturtasten, gewölbtes Tastenfeld, numerischer Tastenblock und integrierte LCD-Anzeige. Das eigentliche Handicap, nämlich die unzweckmäßige QWERTY- bzw. QWERTZ-Anordnung, ist aber anscheinend nicht auszurotten, weil niemand umlernen will. Solche gefühlsmäßigen Abneigungen bilden oft das entscheidende Hindernis beim Versuch, technische Geräte besser an die menschlichen Bedürfnisse anzupassen.

Durch das Vordringen der Computer am Arbeitsplatz und im privaten Bereich rückt die Ergonomie für viele ins Blickfeld. Farbliche und akustische Effekte beispielsweise können die Arbeitsbereitschaft stark beeinflussen: Bei der Telefonvermittlung wird wartenden Anrufern mancherorts Musik zur Besänftigung vorgespielt, und ein grüner Bildschirm wird zumeist als angenehm empfunden, während sich beim Arbeiten mit den alten Schwarzweißschirmen nach ein paar Stunden Streßsymptome einstellen.

Es ist eine alte Erfahrung der Ergonomie, daß Mißvergnügen an der Umgebung (oft ohne bewußtes Erkennen der direkten Ursache) die Anfälligkeit für Seh- und Rückenbeschwerden erhöht und einen erhöhten Ausfall durch Krankheit zur Folge hat. Rechnerbenutzer leiden oft unter den Raumverhältnissen oder einer nicht durchdachten Tätigkeitsgestaltung. Bei intensiver ergonomischer Untersuchung eines Büros mit vielen Bildschirm-Terminals, die einfach längs der Wand aufgestellt worden waren, erwiesen sich Streß- und Isolationsgefühle, die durch diese Anordnung bedingt waren, als Ursache für die niedrige Effizienz und die hohe Fehlerquote. Nachdem die Arbeitnehmer in der Raummitte an gegeneinander gestellten Tischen mit niedrigen Aufbauten untergebracht waren und somit eine freundlichere Atmosphäre geschaffen worden war, wurde die Arbeitsqualität nun sofort bedeutend besser.

Die Einführung von Computern hat häufig zu einer Tätigkeitsabwertung geführt, weil schematische und reizlose Arbeit die Menschen zu wenig forderte. Heute gehört es bei einer Umstellung auf Datenverarbeitung zu den selbstverständlichen Pflichten des Systemanalytikers, eine hinreichend anspruchsvolle und befriedigende Beschäftigung sicherzustellen. Das erfordert sachverständige ergonomische Beratung.

Bei einem guten ergonomischen Konzept wird der arbeitende Mensch als wesentliche Systemkomponente mit spezifischem Verhalten betrachtet. Humanitäre und betriebswirtschaftliche Überlegungen gebieten gleichermaßen, daß das Wohlbefinden der Mitarbeiter im Rang nicht hinter Adreßbuskapazität und Taktfrequenz liegen darf.

Auch der Programmierer kann aus einer sorgfältigen Analyse der besonderen Anforderungen und Probleme bei seiner Arbeit Nutzen ziehen. Ständige Konzentration, logisches Vorgehen und Akribie liegen von Natur aus den wenigsten – die meisten Menschen wehren sich sogar unbewußt gegen solche Auflagen. Das ist eine der Ursachen für die unvermeidlichen Programmierfehler.

Die Gestaltung neuer Programmiersprachen oder Betriebssysteme unter ergonomischen Aspekten ist äußerst reizvoll und bietet dem Anwender viele Vorteile. Psychologen haben den Menschen als „informationsverarbeitendes System" ausgiebig studiert, insbesondere Gedächtnisstruktur, Aufnahmefähigkeit und Zeitverhalten. Diese Erkenntnisse unterstützen die Entwicklung benutzerfreundlicher Systeme, die ohne viel Übung wie selbstverständlich zu handhaben sind.

Bereits seit längerer Zeit werden psychologische Überlegungen in der Industrie bei Trainigs-, Auswahl- und Organisationsmethoden berücksichtigt. Beispielsweise kann man auf Konstrukteure so einwirken, daß sie ihren Entwurf anwendergerecht gestalten, statt vorauszusetzen, der Benutzer werde sich schon an das neue System gewöhnen.

Neuerdings befaßt sich die Ergonomie verstärkt mit der „Schnittstelle Benutzer", das heißt mit der Bedienung komplexer Softwaresysteme. Früher waren die Benutzer fast ausschließlich versierte, stark motivierte Profis, die für eine hohe Rechnerleistung ohne weiteres Unbequemlichkeiten und den Zwang zur Aneignung spezifischer Kenntnisse akzeptierten. Der heutige Benutzer ist dagegen ein Durchschnittsmensch mit begrenzter Nachsicht gegenüber anspruchsvollen und „eigenwilligen" Apparaten. Und nicht nur Gelegenheitsbenutzer haben Probleme mit moderner Technologie: Zahlreiche Datenbanken, die megabyteweise Management-Information zur Verfügung stellen können, werden falsch oder gar nicht genutzt, weil dafür die Beherrschung komplexer Sprachen wie SQL Voraussetzung ist. Entwickler wie Anwender setzen auf die künftigen „Umgangssprachen"-Terminals, die hoffentlich auch eine Kommunikation mit Datenbanken und Finanzprogrammen im Alltagsdeutsch zulassen. Dabei muß das Terminal alle Eingaben in Rechnerbefehle umwandeln und umgekehrt die Antworten des Systems in verständlicher Form ausgeben.

Programme erklären sich

Es gibt viele Wege, dem Benutzer die Arbeit zu erleichtern, etwa mit HELP-Routinen. Forschungsarbeiten über „künstliche Intelligenz" haben zur Entwicklung von erfahrungsorientierten Programmen geführt, die ihre Entscheidungsfindung selbst erklären. Ähnlich könnte man zu intelligenten Beratungs-Modulen für Softwarepakete kommen, allerdings nicht unter Umgehung der Frage, wann und für wen eine Erklärung wichtig ist. Während den Programmierer Datenstrukturen und Verarbeitungsprozesse interessieren, möchten kommerzielle Anwender eigentlich nur wissen, wie man mit der Software etwas geschäftlich Verwertbares zuwege bringt.

Eine Rolle spielen bei der „Schnittstelle Benutzer" sicher auch klischeehafte Vorstellungen. Viele gehen mit Vorurteilen an Computer heran, wobei der Rechner oft für gescheiter gehalten wird als ein Mensch, der die gleiche Arbeit verrichtet. Oder der Rechner gilt wegen der knappen Kommentare vieler Sofwarepakete (speziell bei Eingabefehlern) als unfreundlich, sogar feindselig – eine unsinnige Vermenschlichung des Apparats, die die Arbeit unter Umständen belasten kann.

Unüberlegte Methoden zur Hebung der Benutzerfreundlichkeit, die den Computer intelligenter erscheinen lassen als er ist, machen das nur schlimmer. Launige Dialogformeln wie HALLO FRITZ, ICH BIN DER GUTE GEIST DER DATENBANK ermuntern den Benutzer geradezu, im gleichen Stil zu antworten – im allgemeinen mit niederschmetterndem Ergebnis und entsprechendem Frustrationseffekt.

Echte Benutzerfreundlichkeit erreicht man nur mit viel Geschick: Man muß sich in die Situation des Benutzers einfühlen und versuchen, sein Verhältnis zu seiner Tätigkeit und zum Rechner zu deuten. Ergonomische Gestaltung zeigt sich nicht darin, daß Sie den Bildschirm schräg stellen und die Plattenlaufwerke violett anmalen!

Es ist allgemein bekannt, daß die Leistungsfähigkeit des arbeitenden Menschen durch grobphysiologische Einflußfaktoren wie Tischhöhe, Lufttemperatur oder Geräuschpegel beeinträchtigt werden kann. Aber auch die mittelbaren Effekte von Farbgebung, Lichtwirkung und Raumgefühl gelten inzwischen als gleich wichtig bei der Beurteilung von Arbeitssystemen, insbesondere in der Datenverarbeitung. Neben der Auswahl von Hardware und Software muß der Systemanalytiker, der eine neue EDV-Anlage aufstellt, eine ganze Reihe derartiger Gesichtspunkte berücksichtigen.

Flexibel in der Anwendung

Festgelegte Denkmodelle können genauso frustrierend und beengend wirken wie eine ungemütliche Umgebung. Die Sprache FORTH, die vielerorts als künftiger BASIC-Ersatz angesehen wird, hat der Astronom Charles Moore entwickelt. Er arbeitete am Kitts Peak-Observatorium in Arizona an der Steuerung von Teleskopbewegungen und merkte bald, wie wenig FORTRAN als rein datenverarbeitende Sprache für Steuerungszwecke brauchbar ist. Deshalb schuf er mit FORTH etwas Neues, was sich von starr strukturierten Sprachen wie FORTRAN dadurch unterscheidet, daß für jede spezielle Programmieraufgabe praktisch ein neuer Dialekt entwickelt werden kann. Als Preis für die hohe Flexibilität ist FORTH nicht leicht erlernbar – effiziente Systeme sind nicht immer bequem in der Handhabung.

Die Planung

Wir beschäftigen uns mit der Programmgestaltung und überlegen, was zu berücksichtigen ist, bevor man überhaupt mit dem Schreiben eines Listings beginnt.

Die Programmgestaltung wird von den Beteiligten – dem Designer/Programmierer und dem Anwender – als eine wichtige Übung zur angewandten Problemlösung betrachtet. Häufig geht man davon aus, daß die zu lösenden Probleme vornehmlich technischer Art sind – wie den Bildschirm zu formatieren, die Ausführungsgeschwindigkeit zu erhöhen und so weiter. Tatsächlich aber beginnen die Probleme schon mit der Formulierung der eigentlichen Aufgabe des Programms. Die Grundlagen zu ihrer Lösung werden gewöhnlich im ersten Gespräch zwischen dem Anwender und dem „Experten" geschaffen. Anwender sind sich jedoch selten über die wahre Natur der Probleme im klaren. Sie hoffen, vom Experten eine Problemanalyse und eine Lösung zu bekommen. Und Experten glauben oft, ein Problem zu kennen – und auch die Lösung –, bevor der Anwender dieses überhaupt erklärt. Durch diese Kommunikationsschwierigkeiten kommt es zu einer unvollständigen Problembeschreibung, und die Belange des Anwenders werden ungenügend berücksichtigt. Auf dieser Grundlage ausgeführte Arbeiten haben als Ergebnis ein für den Anwender unbefriedigendes Produkt, das er nur widerwillig akzeptiert.

Kommunikationsprobleme

Bei Heimcomputer-Projekten ist der Programmierer meistens Designer (oder „Systemanalytiker") und Anwender in einer Person. Man sollte annehmen, daß die Kommunikationsprobleme nicht auftauchen. Trotzdem kann es zu denselben Schwierigkeiten kommen, wenn man sich nicht selbst die Probleme, Lösungen und Erfordernisse so darlegt, als spräche man mit einer anderen Person darüber.

Beschäftigen wir uns mit einem imaginären Anwender und seinem Problem: Er ist ein guter Flugzeugmodellbauer, der einen Computer mit Cassettenspeicher hat. Er will detaillierte Beschreibungen der beim Bau verwendeten Materialien all seiner Modelle speichern, um bei späteren Bauten nachsehen zu können, welcher Leim und welche Verbindungen eingesetzt wurden. Der Programmierer benötigt deshalb vom Anwender folgende Anweisungen:

- Programmfunktion: Dies kann eine vage Absichtserklärung sein wie „Es sollte meine Modellbau-Aufzeichnungen speichern können", die dann aber durch Nachfragen und Überlegen des Programmierers weiter spezifiziert wird. Zum Beispiel: „Es sollte meine Modellbeschreibungen, seine Konstruktionen und die dabei verwendeten Materialien so speichern, wie ich sie über Tastatur eingebe, und sie darstellen, wenn ich die Modellbezeichnung oder ein bestimmtes Konstruktionsmerkmal eingebe." Das macht die Bedürfnisse des Anwenders deutlicher und verweist auf bestimmte Programmieraufgaben (Speichern, Suchen, Inhaltsverzeichnis, Zugriff etc.).

Genaue Beschreibung erwünscht

- Programmnutzung: Einige der formellen Einzelheiten der gewünschten Nutzung mögen zwar aus der Funktionsbeschreibung deutlich werden, sind aber eventuell nicht vollständig. Vielleicht möchte der Anwender beispielsweise nicht, daß die Daten auf dem Bildschirm dargestellt werden, sondern wünscht lediglich eine exakte Druckerausgabe.
- Gestaltung des Programms: Ein- und Ausgabe-Formate. Professionelle Programmierer benutzen häufig vorgedruckte Karten, worauf jeder Screen so dargestellt ist, wie ihn der Anwender in den Eingabe/Ausgabe-Phasen sieht oder sehen möchte. Bildschirm-Formate sind wichtige Eigenschaften der sogenannten „Benutzer-Schnittstelle", womit die Zusammenarbeit von Anwender und Maschine gemeint ist.
- Programmorganisation: Datei- und Programm-Formate. Der Anwender möchte vielleicht mindestens 100 Flugzeugmodell-Beschreibungen speichern. Die Größe der Programm-Datenfiles hat erheblichen Einfluß auf ihr Format und die Zugriffsmöglichkeiten. Der Suchlauf durch beispielsweise sechs Modellbeschreibungen auf Cassette beansprucht circa fünf Minuten, was für den Benutzer zumutbar ist. Auf das Durchsuchen von 100 Modellen zu warten wäre dagegen indiskutabel. Eine Lösungsmöglichkeit bestünde darin, Programm und Index-File auf einer Cassette zu speichern, die Beschreibungen dagegen auf 20 anderen Cassetten abzulegen, beispielsweise klassifiziert nach Flugzeugtypen.
- Aufgabenstellung: spezielle Prozeduren und Berechnungen. Bei unserem Beispiel mag sich dieses Problem nicht stellen. Doch es wird deutlich, wenn man andere Probleme mit berücksichtigt. Es mag 20 perfekte und gleichermaßen befriedigende Wege zur Lösung eines

bestimmten Problems geben. Der Anwender aber verlangt nur nach einem bestimmten.

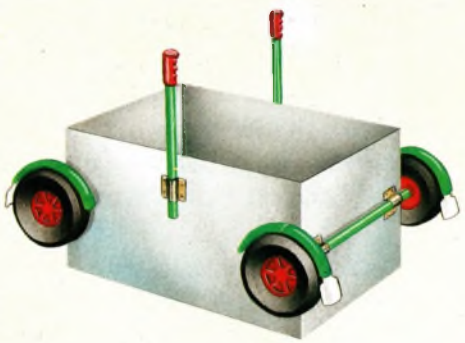
Berücksichtigt man dies nicht, wird der Benutzer mit dem Programm unzufrieden sein. Der Autor mag sich von der vom Anwender bevorzugten Methode wegen besserer Wirtschaftlichkeit anderer Methoden entfernt haben. Der sich daraus ergebende Vorteil aber erweist sich als null und nichtig, wenn der Anwender ein solches Programm nicht benutzen will. Die Arbeitsweise des Anwenders zu verstehen kann bei der Programmberechnung nützlich sein.

Liegen all diese Informationen vor, kann man mit der Umwandlung der Einzelheiten in ein Programm beginnen. Als sinnvoll hat sich erwiesen, zunächst den Anwender-Programm-Dialog zu schreiben, dann die Daten-Files und schließlich die Steuerprozesse. Mit dem Wort „Dialog" ist die zweiseitige Kommunikation, der Austausch von Informationen zwischen Anwender und Programm, gemeint. Diese besteht bei unserem Beispiel nicht allein aus der Eingabe von Modellflugzeug-Einzelheiten und ihrer richtigen Darstellung, sondern schließt auch jedes Prompt, Menü und jede Nachricht ein, die das Programm erzeugen kann, sowie alle Eingaben oder Befehle, die der Benutzer gibt. Die Festlegung der Art, wie der Dialog erfolgen soll, ist in diesem Stadium ebenfalls wichtig. Für das Flugzeugprogramm könnte eine Mischung zwischen Menü und befehlsgebundener Interaktion geeignet sein. Die hier getroffene Entscheidung beeinflußt die Struktur des gesamten Programms. Inhalte und Format des Dialogs müssen in allen Einzelheiten berücksichtigt werden. Die „Belohnung" für all diese Mühen ist, daß die vom Programm verarbeiteten Daten nun auch noch spezifiziert werden müssen. Danach kann der für Fehlermeldungen und Bestätigungen erforderliche Speicherplatz berechnet werden, und die Files können programmiert werden. Für das Flugzeugmodell-Programm, dessen Files große und lange Textblöcke enthalten, mag die Untergliederung in mehrere Unterprogramme auf verschiedenen Cassetten die beste Lösung sein.

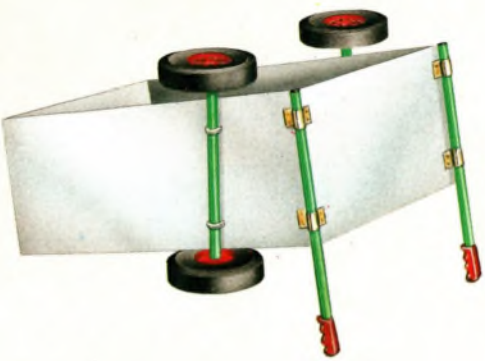
Angepaßte Funktionen

Die erforderlichen Funktionen werden jetzt offenkundig. So muß es Routinen geben, die das Hinzufügen und Editieren von Text-Daten erlauben, ferner das Speichern neu eingegebener Texte (womit automatisch das Inhaltsverzeichnis aktualisiert werden muß), die Suche und Darstellung von Beschreibungen muß möglich sein usw. All dies sollte dem Anwender in Form von Optionen angeboten werden.

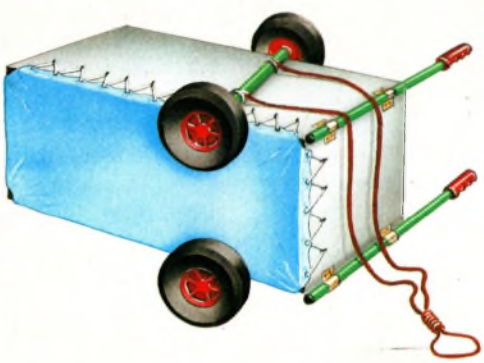
An diesem Punkt sollte der Benutzer die Programmgestaltung sorgfältig überprüfen, um festzustellen, ob es wirklich allen Erfordernissen genügt. Erst dann sollte das Programm codiert werden. Das ist natürlich leichter gesagt als getan – doch dazu später Genaueres.



Der Designer glaubte, der Anwender wolle dies



Der Programmierer vermutete, dies meine der Designer



Das wurde dem Anwender schließlich verkauft

Und das wollte der Anwender eigentlich haben

Wenn das typische Software-Entwicklungsteam – bestehend aus Anwender, Gestalter und Programmierer – das Problem zu lösen hätte, wie man schwere Lasten im Garten bewegt, könnte dies dabei herauskommen. Kommunikationsschwierigkeiten zwischen Experten und Laien und Experten untereinander sind noch immer Kernproblem aller Entwicklungsteams.

Algorithmen

Es wird untersucht, wie sich die Programmerstellung mit Hilfe von Algorithmen optimieren läßt.

Algorithmen können Abläufe entweder durch „einfache" Vorgänge beschreiben, die keiner Erklärung bedürfen, oder mit anderen, zuvor definierten Vorgängen. So genügt im Rezept eines Kochbuches die Anweisung „bereiten Sie eine Bechamelsoße", wenn das Rezept für Bechamelsoße an einer anderen Stelle des Buches ausführlich erklärt ist. Programmalgorithmen bestehen daher aus Befehlen, die wiederum Algorithmen (Prozeduren, Routinen, Funktionen) einsetzen. Diese (Unter-)Algorithmen sind entweder an anderen Stellen des Programms bereits angelegt oder in der Sprache selbst enthalten (beispielsweise Befehle wie PRINT und DIM oder Funktionen wie LOG und TAN).

In diesem Artikel untersuchen wir, wie sich Algorithmen aus anderen Algorithmen und „einfachen" Abläufen (den Befehlen und Funktionen einer Programmiersprache) aufbauen lassen. Zunächst werden die Algorithmen für grundlegende Aufgaben entwickelt (beispielsweise die Bewegung eines Sprites oder eine Zahleneingabe). Die Grundalgorithmen werden dann zu umfassenderen zusammengesetzt, die etwa den Bildschirm aufbauen oder ein Menüsystem steuern. Diese Algorithmen sind wiederum Teile größerer Funktionszusammenhänge, so daß sich am Ende das gesamte Programm als ein Algorithmus mit vielen untergeordneten Algorithmen ansehen läßt. Dieses Konzept bildet die Grundlage der „strukturierten" oder „modularen" Programmierung, auf die wir später genauer eingehen.

Jeder Algorithmus besitzt eine Eingabe und eine Ausgabe. Mit anderen Worten, Algorithmen sind Vorgänge, die Eingangsdaten in Ergebnisdaten umwandeln. Die Eingangsdaten werden dem Algorithmus von außen in Form von „Parametern" übergeben. Sie erscheinen dem Algorithmus als Konstanten, können sich aber zwischen zwei Aufrufen eines Algorithmus verändern. Parameterübergaben sind schon dem Anfänger vertraut. Das einfache Programm:

```
10 PRINT "HALLO!"
```

übergibt den Parameter „HALLO!" an den Algorithmus, den der Befehl „PRINT" aufruft. Ähnliche Beispiele sind TAN(P), LEFT$ (P$,5) oder POKE P,5, wobei P, P$ und 5 die Parameter sind. Die Ergebniswerte des Algorithmus sind ebenfalls Parameter. Programmiersprachen mit lokalen Variablen (wie PASCAL und C) übergeben die Parameter im allgemeinen gleich beim Aufruf einer Prozedur:

```
procedure ( parameter1, parameter2 etc );
```

Beim Aufbau eines Algorithmus besteht der erste Schritt immer darin, den Inhalt, die Typen (Ganzzahl, reale Zahl, String etc.), die Größe und den Wirkungsbereich der Ein- und Ausgabeparameter festzulegen.

Nach den Definitionen der Ein- und Ausgabe wird im nächsten Schritt skizziert, wie die Eingabeparameter in Ausgabeparameter verwandelt werden können. Da es für diese Umformung kein allgemeingültiges Rezept gibt, ist man hier auf seine eigene Kreativität angewiesen. Es gibt jedoch einige systematische Ansätze, die dabei helfen können.

Die einfachste Methode besteht darin, fertige Algorithmen zu übernehmen. Viele Programmiersprachen bieten brauchbare Algorithmen wie String-Verarbeitung, trigonometrische Funktionen, Ein- und Ausgabemechanismen und Module für Sortierung und Matrixbearbeitung. Doch auch Ihre eigenen Programme können einsatzbereite Algorithmen enthalten. Sie brauchen den bestehenden Code dann nur in neue Programme zu übernehmen.

Es lohnt sich weiterhin, die in Computerzeitschriften veröffentlichten Programme auf brauchbare Routinen durchzusehen. Und schließlich gibt es Bereiche, deren Algorithmen zwar nicht für Computer gedacht waren, die sich aber vielseitig einsetzen lassen. Es gibt beispielsweise zahlreiche Bücher über Buchhaltung, die Formeln für die Berechnung von Salden, Abschreibungen etc. enthalten. Mit ein wenig Nachlesen lassen sich Buchhaltungsprogramme so viel einfacher und zuverlässiger entwickeln. Das Prinzip läßt sich auch auf andere Bereiche wie Elektronik oder Mathematik anwenden.

Klar und wirksam

Ob Sie einen Algorithmus nun übernehmen oder von Grund auf entwickeln, jeder Ablauf muß zwei grundlegende Kriterien erfüllen: Er muß klar und er muß wirksam sein. Klarheit schließt mehrdeutige Anweisungen aus, die sich bei der Entwicklung eines Algorithmus leicht einschleichen können. Eine Routine, die aus einer Liste alle mit „A" und „B" anfangenden Namen heraussuchen soll, könnte folgendermaßen aussehen:

```
IF ANFANG='A' AND ANFANG='B' THEN ...
```

Da sich die Wörter „und" und „oder" der Um-

gangssprache grundlegend von dem AND und OR der booleschen Logik unterscheiden, ist diese Programmzeile falsch: Hier wird ein logisches OR gebraucht!

Wirksamkeit bedeutet, daß Programme keine Befehle enthalten, deren Ausführung unmöglich ist. Befehle sind wirksam, wenn sie mit Papier und Bleistift in einem endlichen Zeitraum ausgeführt werden können. Anweisungen wie „setze X auf die höchste Primzahl" sind nicht wirksam, da es keine „höchste Primzahl" gibt.

Bestimmte Kriterien bewerten Algorithmen auch als Ganzes. So müssen Algorithmen beendet werden können. Die folgende Routine erfüllt dieses Kriterium nicht (obwohl ihre Befehle klar und wirksam sind), sondern erzeugt eine Endlosschleife:

Schritt 1 Setze X gleich 1
Schritt 2 if X > 3 then Ende
Schritt 3 gehe zu Schritt 1

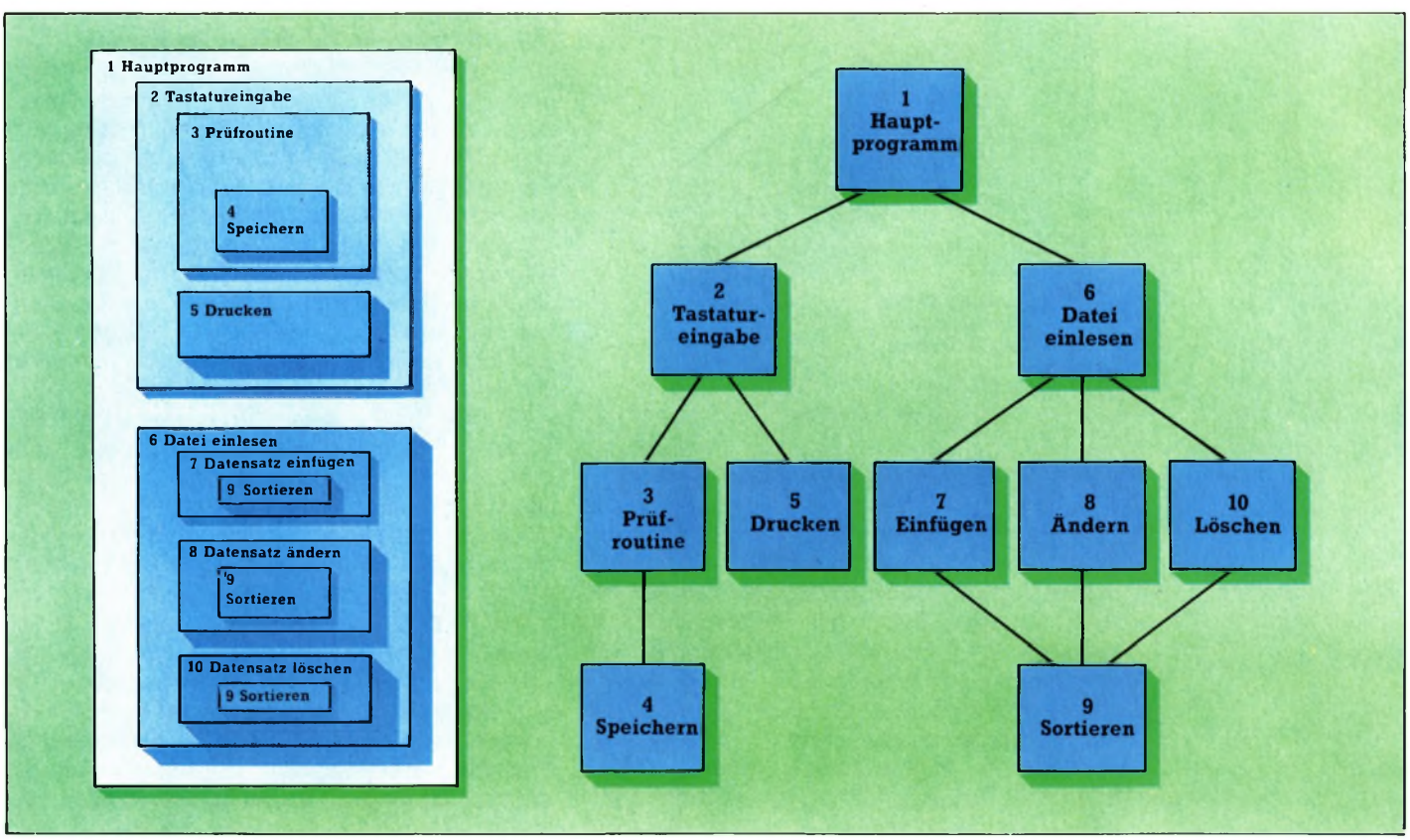
Es läßt sich nicht immer auf eine einfache Weise feststellen, ob ein Algorithmus beendet werden kann. Im allgemeinen enthalten Schleifenalgorithmen Bedingungen, die einen Ausstieg aus der Schleife ermöglichen. Es muß jedoch immer überprüft werden, ob die Endbedingung überhaupt eintreten kann.

Über die Kriterien Wirksamkeit, Allgemeingültigkeit und Eleganz lassen sich Algorithmen miteinander vergleichen. Wirksamkeit wird dabei aufgrund von Schnelligkeit und Speicherbedarf beurteilt. Oft – aber nicht immer – arbeiten diese beiden Komponenten Hand in Hand (ein schneller Code braucht selten viel Speicherplatz). Ist ein Algorithmus einmal gefunden, sollte seine Wirksamkeit durch geeignete Anordnung der Befehle optimiert werden. Beispielsweise lassen sich Berechnungen mit Ganzzahlen weitaus schneller durchführen als mit Fließkommaarithmetik. Zur Optimierung einer Routine müssen daher oft völlig neue Algorithmen gefunden werden, die die gleiche Aufgabe erfüllen.

Allgemeingültig

Allgemeingültigkeit ist ein weiteres wichtiges Merkmal: Ein Algorithmus sollte nicht nur für eine Aufgabe Gültigkeit haben, sondern ein breites Spektrum von Aufgaben ausführen können. Auf lange Sicht gesehen lohnt es sich, Algorithmen so allgemeingültig wie möglich anzulegen. Wenn beispielsweise ein Programm vom Anwender mehrmals eine JA/NEIN-Entscheidung verlangt, sollte die entsprechende Routine die Meldung „Eingabe J(a) oder N(ein)" darstellen, die Eingabe annehmen, überprüfen, ob auch „J" oder „N" eingegeben wurde, und den Anwender entweder erneut zur Eingabe veranlassen oder das Ergebnis als Parameter zurückgeben. Elegante Algorithmen sind zugleich einfach und raffiniert. Wirksame und allgemeingültige Algorithmen lassen sich jedoch leichter aufbauen.

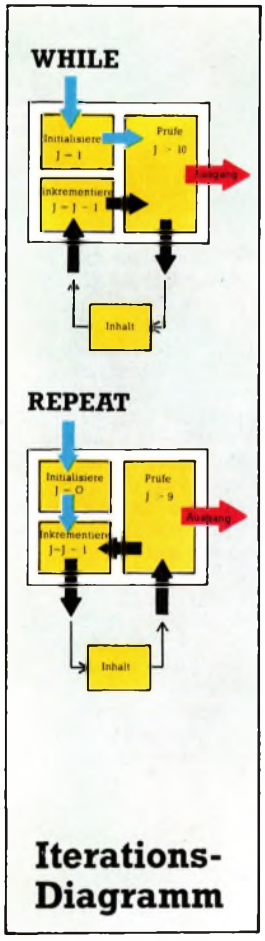
Für Algorithmen sind weiterhin die Steuerung und der Datenfluß wichtig. Wie diese beiden Komponenten in einem Flußdiagramm dargestellt werden, behandeln wir in der nächsten Folge dieser Serie.



Das Blockdiagramm (links) zeigt, wie die Algorithmen eines Programms ineinander verschachtelt sind, während das Flußdiagramm (rechts) die Funktionsebenen der gleichen Vorgänge darstellt. Die einfachsten Algorithmen befinden sich dabei in der größten Schachtelungstiefe beziehungsweise auf der untersten Ebene der Hierarchie.

Schleifenstruktur

Flußdiagramme sind eine wichtige Technik bei der Programmgestaltung, doch die übliche Schreibform ist nicht immer ausreichend, vor allem dann, wenn es sich um komplizierte Schleifenstrukturen handelt.



Iterations-Diagramm

Wiederholung – oder Schleifenbildung – ist eine der grundlegenden Strukturen jeder Programmiersprache. Wir erwähnten bereits an anderer Stelle, daß immer dann im Algorithmus eine Schleife zu verwenden ist, wenn eine Entscheidung den Fluß der Steuerungsdaten umlenken soll.

Da Schleifen eine wichtige Strukturform darstellen und circa 60 Prozent der gesamten Prozessor-Aktivität ausmachen, ist es sehr wichtig, sie einmal detaillierter zu betrachten. Im Mittelpunkt der Betrachtung stehen ihre Einwirkung auf ein grundlegendes Programm, also eine Algorithmus-Struktur, und die verschiedenen Methoden, mit denen sie konstruiert und klassifiziert werden können.

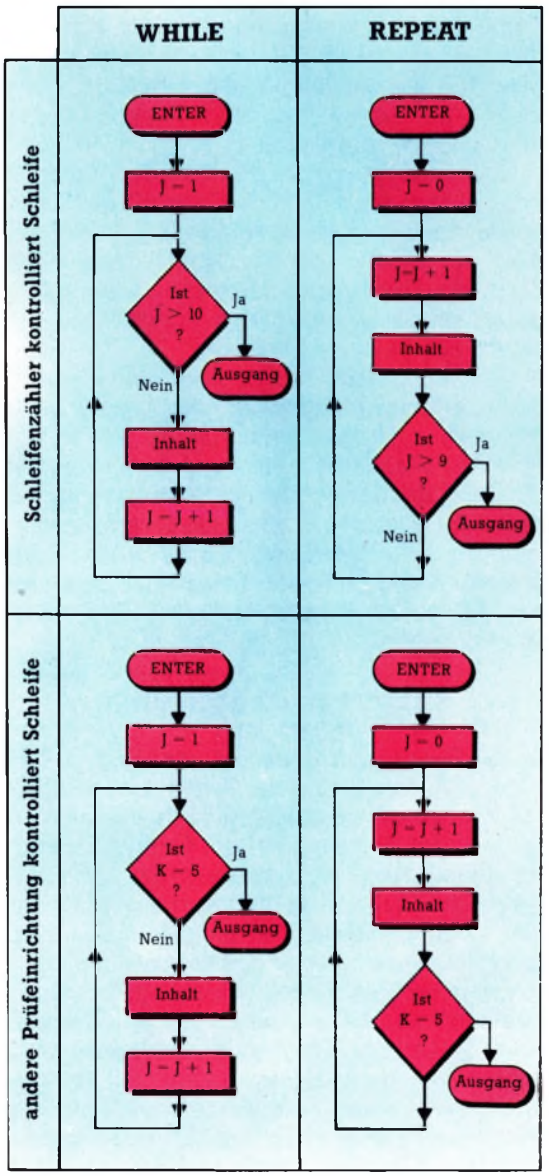
Schleifen werden häufig in zwei Kategorien eingestuft, abhängig von der Ähnlichkeit zu den beiden Hochsprachen-Schleifenstrukturen „REPEAT ... UNTIL" und „WHILE ... ENDWHILE", die beide in PASCAL verwendet werden. Die REPEAT-Schleife ist in zahlreichen BASIC-Versionen vorhanden. Die beiden Strukturen unterscheiden sich in der Position des Schleifenausgang-Tests: Bei einer REPEAT-Schleife erfolgt die Überprüfung am Ende der Schleife, wogegen sie bei einer WHILE-Schleife zu Beginn positioniert ist. Dieser erste Unterschied ist im Fluß-Diagramm deutlich zu sehen.

Eine andere Möglichkeit der Schleifen-Klassifizierung hängt davon ab, ob die Variable, die als Schleifenzähler dient, im Schleifenausgang-Test verwendet wird oder ob ein anderes Element zur Steuerung und Kontrolle des Schleifenendes benutzt wird. In einem Flußdiagramm konventioneller Art ist das nur undeutlich zu erkennen.

Klare Darstellung benötigt

Es gibt jedoch eine deutlichere Darstellungsform, „Iterations-Diagramm" genannt, die den Beginn einer Schleife klar kennzeichnet und die Verwirrung bei der Unterscheidung von Schleifen und Verzweigungen beseitigt. Sie besteht aus drei miteinander verbundenen Kästen. Der erste zeigt die Initialisierung des Zählers. Der zweite dient zum Zählen selbst, wogegen der dritte die Schleife beim Ausgang überprüft. REPEAT- und WHILE-Funktionen können in dieser Form dargestellt werden. Die Datenflußrichtung ist dabei verschieden. Der Test-Kasten zeigt an, ob die Schleifen zählerkontrolliert arbeiten oder nicht. Diese Punkte werden im Diagramm deutlich sichtbar.

In einer REPEAT-Schleife fließt die Kontrollfunktion folgendermaßen: „Initialisiere – Aussage – Prüfe – Aussage –Prüfe". Die WHILE-Schleife läuft so ab: „Initialisiere – Prüfe – Aussage – Prüfe – Aussage". Dies ist an der Richtung der Kontroll-Pfeile des Iterations-Diagramms zu erkennen.



Entscheidungen

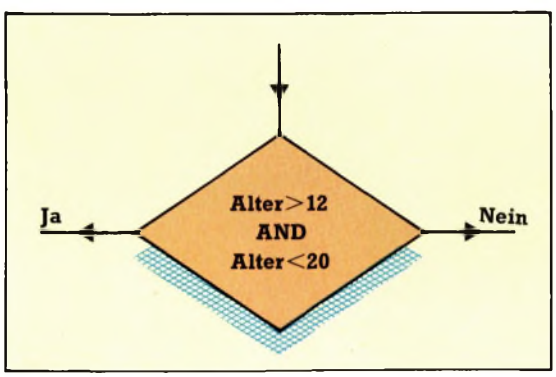
Wir wollen uns noch etwas gründlicher mit der Technik des Flußdiagramms zur Erleichterung der Programmentwicklung befassen. Heute soll gezeigt werden, wie sich ein komplizierter Entscheidungsprozeß in immer kleinere Elemente zerlegen läßt.

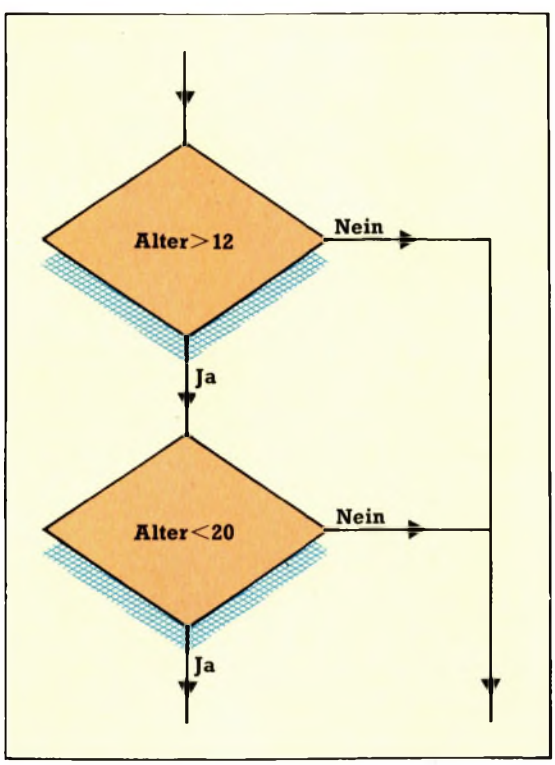
Viele Programmierer verwenden häufig zusammengesetzte Entscheidungen wie:

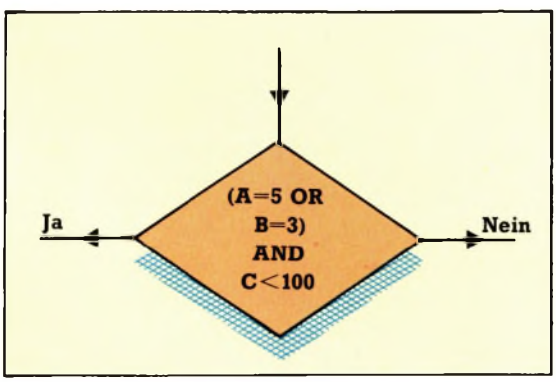
```
IF ALTER>12 AND ALTER<20 THEN
   STATUS="TEENAGER"
```

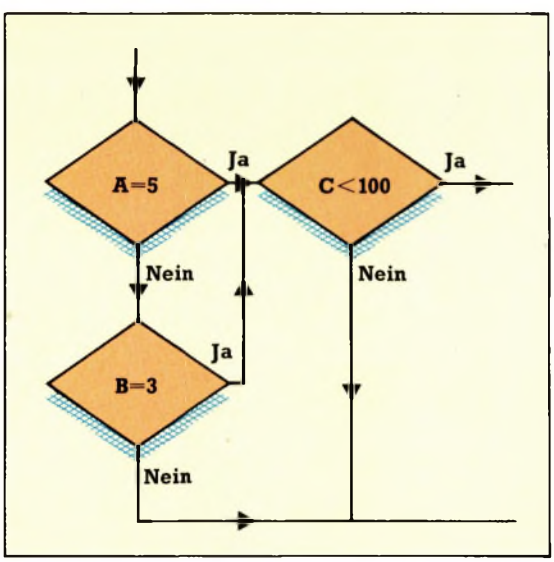
Anweisungen dieser Art sind meist einfacher zu überschauen, wenn die Entscheidungskriterien voneinander gelöst und einzeln betrachtet werden.

Wir haben das BASIC-Beispiel unten in Diagrammform wiedergegeben, um zu zeigen, daß die zusammengesetzte ungünstiger als die einfache Version ist. Das nächste Beispiel besteht aus drei miteinander verbundenen Entscheidungen. Der logische Ablauf im zweiten Diagramm ist dabei wegen der Einzel-Entscheidungen viel übersichtlicher. Es gibt hier eine gewisse Ähnlichkeit mit der Booleschen Logik, die ebenfalls komplizierte Schaltungen aus einfachen logischen Gattern aufbaut.



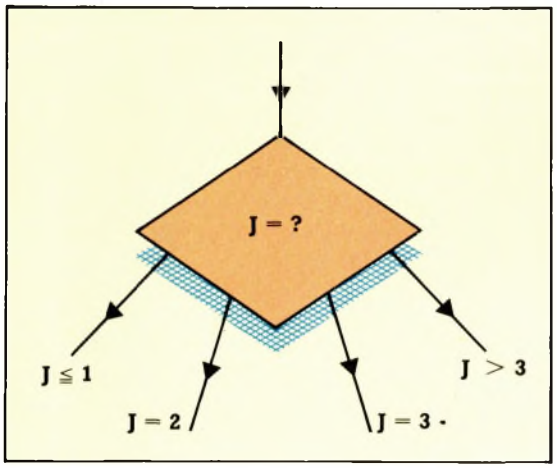


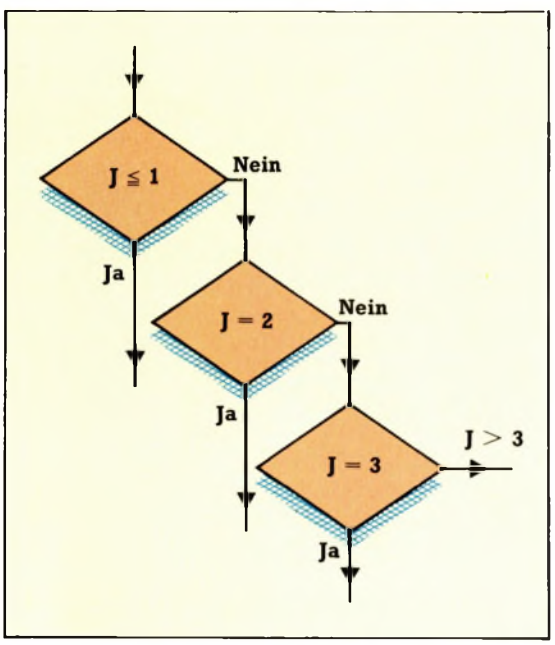




In den Beispielen sind nur Ja Nein-Entscheidungen aufgeführt. Es gibt natürlich auch Fälle, in denen mehr als zwei Möglichkeiten für die Beantwortung einer Frage denkbar sind. So kann etwa bei einem menugesteuerten Programm die Tastatur abgefragt werden, so daß der Programmablauf je nach der gedruckten Taste verzweigt. Dazu gibt es in den meisten Programmiersprachen spezielle Befehle wie CASE .OF in PASCAL oder ON .GOTO bzw. ON .GOSUB in BASIC. Die

Regeln sind jedoch überall gleich: Es darf nur jeweils ein Weg aus dem Entscheidungsfeld gewählt werden. Die Beschriftung muß die möglichen Wege eindeutig angeben, und diese müssen alle Möglichkeiten abdecken. Eine kombinierte Entscheidung kann als Verzweigungsfeld mit mehreren Ausgängen gezeichnet werden.





Kombinierte Entscheidungen lassen sich immer durch mehrere Ja/Nein-Entscheidungen ersetzen.

Besonders bei einer Vielzahl kombinierter Entscheidungen kann eine Entscheidungstabelle das Flußdiagramm ersetzen. Eine solche Tabelle besteht aus vier Teilen: Text, der die Bedingungen für die Regelanwendung beschreibt, Text, der angibt, was geschehen soll, ein Raster, das die Bedingungen mit den Regeln verknüpft, und ein zweites Raster, das die Reaktionen mit den Regeln verknüpft. Im Raster „Bedingungen/Regeln" stehen Variablenwerte, im unteren Raster „Reaktionen/Regeln" stehen die entsprechenden Reaktionen. Ein Zahlenwert dient als Eingabeparameter für die

	Regeln							
Bedingungen	1	2	3	4	5	6	7	8
Feuerknopf gedrückt	✓	✓	✓	✓	✓	✓	✓	✓
Schwierigkeitsgrad	1	1	2	2	1	1	2	2
Spielstärke	Anfänger		Anfänger		Geübter	Spieler	Geübter	Spieler
Reaktionen								
Ausweichmanöver				✓	✓		✓	✓
Bombenschutzschilder			✓	✓			✓	✓
Treibstoff vermindern um	1%	1%	2%	2%	2%	2%	4%	4%

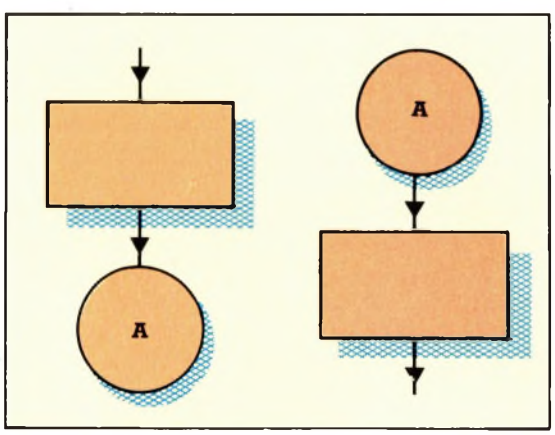
Reaktion. Regel 4 könnte beispielsweise lauten: „Wenn der Feuerknopf gedrückt wird und der Schwierigkeitsgrad 2 von einem Anfänger gespielt wird, dann aktiviere die Bombenschutzschilder und reduziere den Treibstoffvorrat um 2 %".

Mit einer solchen Tabelle können auch mehrere Ja/Nein-Entscheidungen zusammengefaßt werden. Sie entspricht damit genau einer Wahrheitstabelle zur Vorhersage der Ausgabe eines Logikgatters.

Lange Algorithmen zerlegen

Wichtig ist im übrigen, daß sich ein Flußdiagramm nicht über mehr als eine Seite erstrekken sollte – andernfalls verliert man schnell die Übersicht. Zu lange Algorithmen müssen in mehrere Einzelkomponenten zerlegt werden. Dabei kann jeder Algorithmus von einem anderen Algorithmus aufgerufen werden. So läßt sich jede Programmroutine in einen Kasten des Gesamtprogramms hineinschreiben, sogar wenn sie mehrere andere Abläufe aus dem Programm für die eigene Funktion abrufen muß.

Meist geht die Programmentwicklung nicht so glatt, wie man hofft, und das Flußdiagramm muß entsprechend vergrößert werden. In diesem Fall teilt man das Diagramm an einem geeigneten Punkt und verwendet ein Merkzeichen in einem Kreis, mit dem man auf der zweiten Diagrammseite schnell sieht, wo der logische Fluß wieder aufgenommen wird.



Puzzlespiel

Die Technik des modularen Programmaufbaus ist in jeder Programmiersprache nützlich. Sprachen wie PASCAL erleichtern das Arbeiten mit Modulstrukturen, bei BASIC braucht man dafür ein gewisses Durchhaltevermögen. Die Konstruktion von Programmen aus Einzelkomponenten kann hier Hilfestellung geben.

Ein Modul ist eine Befehlsfolge mit einer bestimmten Funktion. Jedes Modul hat exakt definierte Ein- und Ausgänge, seine „Schnittstellen". Was immer zwischen diesen Schnittstellen passiert, sollte vom restlichen Programm vollständig unabhängig sein. Ein komplettes Modul kann als separate Einheit behandelt werden – Daten werden über die Schnittstellen ein- und ausgegeben, was aber im Modul passiert, bleibt verborgen.

Module lassen sich aneinanderreihen, ohne daß der Programmierer sich mit dem „Wie" ihrer Funktion belasten muß. Man kann sehr einfach eine Art Modul-Bibliothek anlegen, aus der man bei Bedarf das Geeignete heraussucht. Und schließlich können Module auch weitergegeben und in fremde Programme eingesetzt werden. Um diese angenehmen Eigenschaften zu nutzen, muß beim Programmieren eines Moduls allerdings sehr genau auf den Datenfluß und den internen Ablauf geachtet werden.

Es gibt eine Grundregel, um sicherzustellen, daß kein Modul „aus der Reihe tanzt" und den Programmablauf stört: Jedes Modul darf nur einen Eingangspunkt und auch nur einen Ausgang haben. Die Befehlsfolge innerhalb des Moduls muß also an einem genau definierten Punkt beginnen und – unabhängig von allen internen Verzweigungen und Schleifen – auch an einem solchen Punkt wieder enden.

Module entsprechen den Algorithmen, die Sie aus den vergangenen Kursabschnitten bereits kennen. In strukturierten Sprachen wie PASCAL können Unterprogramme mit eigenständigen Variablen aufgerufen werden. In diesen Sprachen fällt es relativ leicht, in bestimmte Abläufe (Procedures) zu springen und sie am Ausgangspunkt wieder zu verlassen.

In BASIC läßt sich Ähnliches durch die Kombination von GOSUB...RETURN erreichen: Vom Hauptprogramm wird zum Unterprogramm gesprungen, nach dessen Ablauf geht es mit den vom GOSUB-Befehl angesprochenen Zeilen des Hauptprogramms weiter. Allerdings gibt es keine Einschränkungen in bezug auf die Zeile, die das GOSUB-Kommando anspringt. Zwei verschiedene GOSUBs können etwa zu unterschiedlichen Zeilen im selben Unterprogramm führen und von dort mit nur einem RETURN ins Hauptprogramm zurückspringen. Die Resultate können dabei völlig verschieden sein. Auch die Anzahl der RETURNs ist in BASIC nicht festgelegt.

Ohne Selbstdisziplin geht es in BASIC also nicht. Das heißt: Alle GOSUBs zu einem Unterprogramm müssen zur selben Zeile führen, jedes Unterprogramm sollte nur ein RETURN enthalten. Setzen Sie in die erste Zeile jedes Moduls einen REM-Befehl mit dem Namen des Unterprogramms, und verwenden Sie diese Zeile als Einstiegspunkt. Das RETURN steht in der letzten Programmzeile – das muß nicht sein, schafft aber ein Maximum an Klarheit.

Vorsicht bei GOTO

Der GOTO-Befehl kann Programmstrukturen völlig durcheinanderbringen – verwenden Sie ihn also nur mit Vorsicht! GOTO sollte nur zu Zeilen innerhalb desselben Unterprogramms führen, damit nicht versehentlich ein RETURN übersprungen bzw. auf das falsche RETURN reagiert wird. Manchmal soll ein Unterprogramm vor Ausführung aller Befehle verlassen werden. In diesem Fall sollte GOTO zur letzten Unterprogrammzeile führen.

Besonders fehleranfällig ist die Verwendung von GOTO innerhalb von Programmschleifen. Das unkontrollierte Verlassen der Schleife kann dazu führen, daß der gesamte Rest des Programms als Schleife durchlaufen wird. Also: Innerhalb einer Schleife kein GOTO, das aus der Schleife herausführt. Wenn die Schleife vorzeitig verlassen werden soll, muß der Zählindex der Schleife auf seinen Endwert gesetzt werden und GOTO zur Prüfzeile führen (Zeile mit NEXT bzw. WHILE). Man schreibt WHILE und NEXT daher am besten in eine separate Zeile. Programmstrukturen ohne jedes GOTO sind am übersichtlichsten.

Programmabläufe sind speziell bei Verzweigungen schwer kontrollierbar. Eine Entscheidung sollte daher niemals aus einem Unterprogramm herausführen – es sei denn zu einem anderen klar definierten Unterprogramm. Unabhängig von der Verzweigung muß ein Ablauf immer zum festgelegten Endpunkt des Programmteils führen. Das läßt sich am besten anhand eines Flußdiagramms überprüfen. Oft kann die Verwendung von GOTOs in Programmteilen mit Schleifen und Verzweigungen

durch Flags vermieden werden.

Der Datenfluß in ein Modul hinein und wieder heraus geht wie bei den Algorithmen vor sich. Damit die Module unabhängig voneinander arbeiten, darf außer der Datenübertragung zwischen ihnen keine weitere Beeinflussung auftreten. Vom Hauptprogramm werden die Daten zu einem Modul geschickt, von dort sollen die Resultate wieder zum Hauptprogramm transferiert werden.

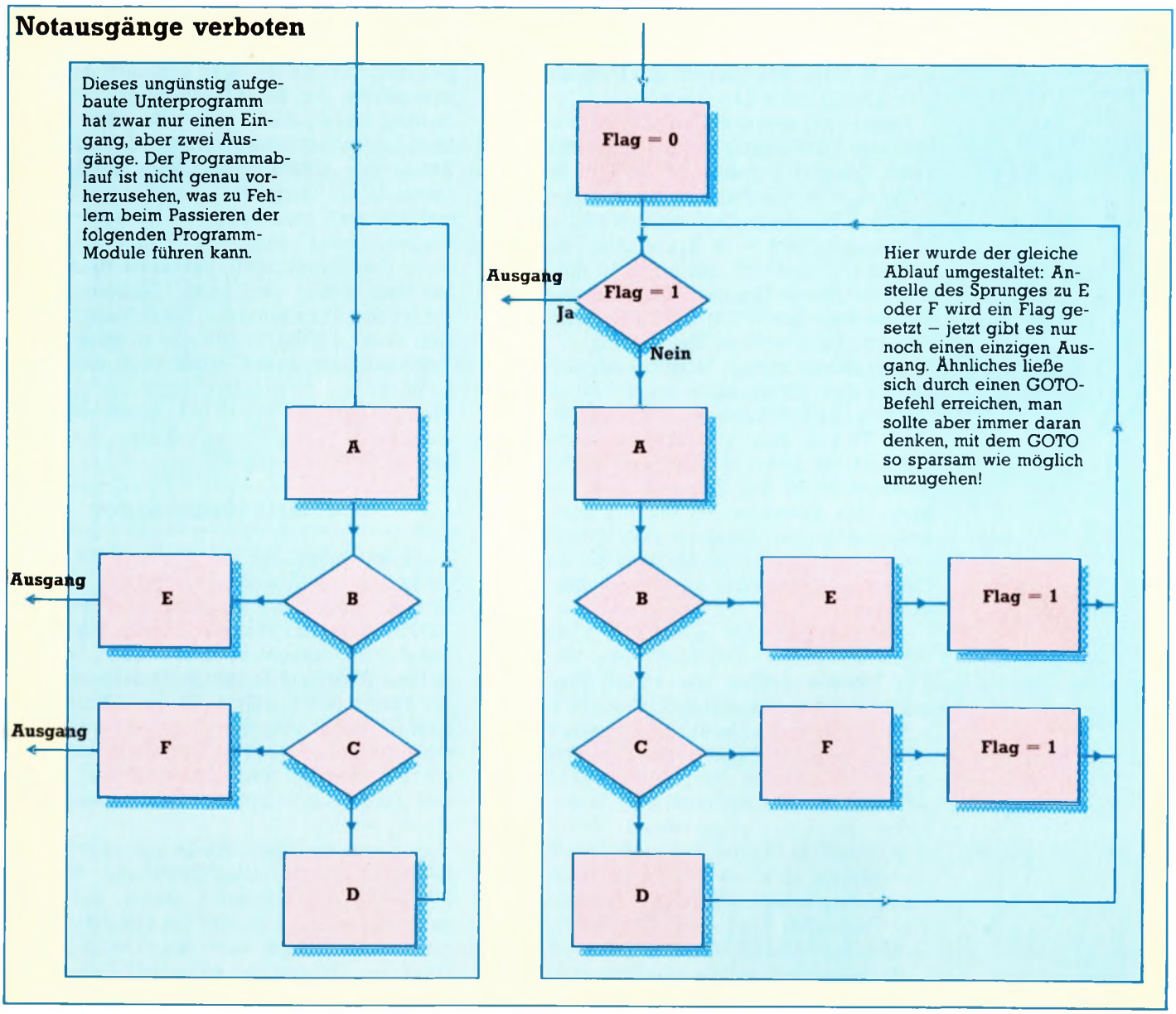
Der Datentransfer innerhalb eines Programms wird über Variablen vorgenommen, deren Bereich („Scope") sich in manchen Programmiersprachen auf Unterprogramme einschränken läßt. In PASCAL müssen etwa die in einem Unterprogramm („Procedure") gültigen Variablen benannt werden. Die „globalen" Variablen des Hauptprogramms gelten überall, während die im Unterprogramm eingesetzten „lokalen" Variablen nur in ihrem speziellen Programmteil Gültigkeit haben.

Lokale Variablen können den gleichen Namen wie globale Variablen haben, ohne daß sich beide gegenseitig beeinflussen. In Programmiersprachen mit lokalen Variablen muß man sich also nicht um differenzierte Benennungen oder die Störung der Variablen durch Variablen aus anderen Programmteilen sorgen. Leider gibt es nur wenige BASIC-Versionen mit lokalen Variablen – wir müssen sie simulieren.

Der einfachste Weg dazu sind Vereinbarungen bezüglich der Variablennamen. Weit verbreitet ist beispielsweise die Verwendung von I, J und K als Schleifenzähler – eine Benennung, die von den Mathematikern übernommen wurde.

Nach der Beschreibung eines Programms in Form eines Flußdiagramms sollte man die einzelnen Unterprogramme numerieren oder auf andere Art benennen. Falls dann Variablen innerhalb eines Unterprogramms lokal eingesetzt werden sollen, hängt man den Namen des Unterprogramms an den Variablennamen.

Notausgänge verboten



Fehlersuche

Methoden zum Erkennen und Verbessern von Fehlern sind für die Programmentwicklung unverzichtbar. Schon durch einfache Tippfehler kann das Programm abstürzen. Ebenso ernste Folgen können jedoch durch versteckte Mängel in der Logik oder beim Programmaufbau verursacht werden.

Fehlerquellen finden sich auf jeder Stufe der Programmentwicklung, bei der Festlegung der Programmeigenschaften ebenso wie bei der Ausarbeitung, der Eingabe oder auch beim Prüfen eines Programms. Schon die Vorüberlegungen bereiten manchmal einem späteren Fehler den Weg. Wenn der Einblick in das gestellte Problem fehlt, entstehen unter Umständen Programme, die etwas völlig anderes tun als beabsichtigt. Hier können die Methoden der strukturierten Programmierung Abhilfe schaffen. Weitere Mängel schleichen sich oft bei der Übertragung der Programmstruktur in die Programmsprache ein. Schon ein falsch getippter Variablenname hat verheerende Folgen. Und auch die Testphase ist nicht ungefährlich. Beim Verbessern eines Fehlers kann leicht ein neuer erzeugt werden.

Die meisten Programmfehler treten allerdings an den „Schnittstellen" auf, entweder zwischen zwei einzelnen Programmteilen oder auch zwischen Programm und Anwender. Beim Passieren einer solchen Schnittstelle sollten alle Werte darauf geprüft werden, ob sie der für das Programm geeigneten Daten-Kategorie angehören. Die Werte können entweder vor der Ausgabe oder im empfangenden Programmteil geprüft werden.

Um sicherzustellen, daß die von einem Unterprogramm weitergegebenen Daten im richtigen Bereich liegen und dem gewünschten Datentyp entsprechen, muß nach ihrer Herkunft gefragt werden. Stammen sie aus einem Datenfile oder aus einer Tastatureingabe? Im Gegensatz zu einem Programm arbeitet ein Mensch nicht immer methodisch und hat zahlreiche Möglichkeiten, falsche bzw. unangebrachte Werte einzugeben. Aber auch Datenfiles können zerstört bzw. fehlerhaft eingelesen werden. Genaue Tests sind daher auf jeden Fall unumgänglich.

Nicht immer stürzt ein Programm durch einen Fehler gleich ab. Wenn es doch geschieht, ist meist eine Regel der Programmiersprache (etwa die unerlaubte Verwendung eines Operators bei RESULT=FIRST$ + SECOND$) oder des Betriebssystems (etwa zuviele gleichzeitig offene Files) nicht beachtet worden. Auf den ersten Blick wirkt dieses Programm völlig korrekt:

```
10 FOR COUNTER = 1 TO 10
20 SUM = SUM + 1
30 PRINT COUNTER,SUM
40 GOTO 10
50 NEXT COUNTER
```

In Wahrheit führt das Programm jedoch in eine Schleife, die einen Fehler auslöst: BASIC verwendet zur Steuerung der FOR. . .NEXT-Schleifen ein Stapelregister, das zu Beginn jeder Schleife um einen bestimmten Wert (in diesem Fall 1) erhöht wird. In diesem Programm wird Zeile 50 mit dem NEXT-Befehl nie erreicht. Das Stapelregister (Stack) wächst immer weiter an – bis zum Programmabbruch durch „Stack Overflow". Fehler dieser Art sind meist leicht zu beheben. Sie sind allerdings nur mühsam zu finden, wenn sie in selten gebrauchten Programmteilen vorkommen. Für die Entdeckung ist dann eine sehr umfangreiche Suche nötig.

Strukturierte Fehlersuche

Fehler werden durch logisches Vorgehen vermieden bzw. schneller gefunden. Dieser Fragenkatalog ist ein Abriß der strukturierten Fehlersuche.

Variablen

1. Sind alle Variablennamen eindeutig? Haben Sie daran gedacht, daß manche Interpreter nur die zwei ersten Buchstaben des Variablennamens lesen?
2. Sind Variablen (speziell Schleifenzähler und Parameter) mehrfach verwendet worden, obwohl sie noch definierte Werte enthalten?
3. Sind die Array-Indizes ganzzahlig? Liegen sie in den erlaubten Grenzen?
4. Beginnen die Array-Indizes beim Element 0 oder beim Element 1?

Berechnungen

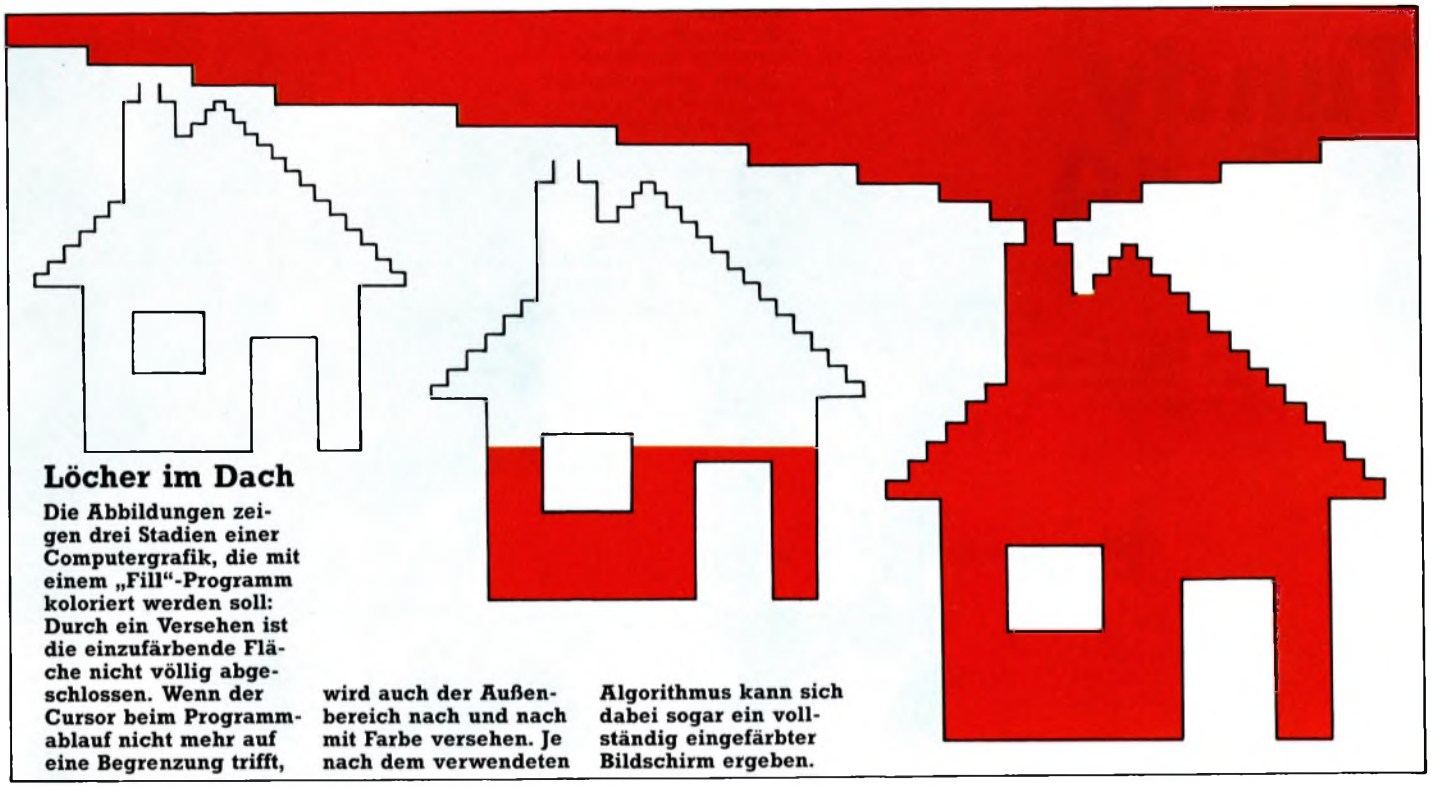
1. Ergeben Berechnungen Strings oder numerische Resultate? Sind die Ergebnisse String- oder numerischen Variablen zugeordnet?
2. Ergibt eine Berechnung Zahlen, die für Computerberechnungen zu groß oder zu klein sind? Kann das zu einer Division durch Null führen?
3. Sind Rundungsfehler von Bedeutung?
4. Werden alle Operationen in der richtigen logischen Reihenfolge ausgeführt?

Vergleichsoperationen

1. Werden Strings nur mit Strings und Ziffern nur mit Ziffern verglichen?
2. Kommt es bei den Strings auf teilweise bzw. vollständige Groß-/Kleinschreibung an?
3. Werden Strings von unterschiedlicher Länge verglichen? Spielt beim Vergleich der Unterschied zwischen den Zeichen oder die Differenz der Stringlänge die wichtigere Rolle?
4. Sind Boolesche- und Vergleichsoperatoren richtig zusammengestellt? A > B OR C beispielsweise ist ungleich A > B OR A > C!

Programmstruktur

1. Begrenzen Schleifen und Algorithmen den Wert irgendwelcher Variablen?
2. Hat jede Schleife und jede Routine nur einen Ein- und einen Ausgang?
3. Geht der Programmablauf nach einem unzutreffenden IF. . .THEN-Befehl zum nächsten Befehl oder in die nächste Programmzeile?
4. Was passiert, wenn bei einer Vielfach-Verzweigung keine einzige der Testbedingungen erfüllt ist?



Löcher im Dach

Die Abbildungen zeigen drei Stadien einer Computergrafik, die mit einem „Fill"-Programm koloriert werden soll: Durch ein Versehen ist die einzufärbende Fläche nicht völlig abgeschlossen. Wenn der Cursor beim Programmablauf nicht mehr auf eine Begrenzung trifft, wird auch der Außenbereich nach und nach mit Farbe versehen. Je nach dem verwendeten Algorithmus kann sich dabei sogar ein vollständig eingefärbter Bildschirm ergeben.

Besondes heimtückisch sind Fehler, die nur die Resultate eines Programms verfälschen, das ansonsten einwandfrei läuft. Als Beispiel haben wir ein „Fill"-Programm gewählt, das auf den Bildschirm gezeichnete Umrisse in einer bestimmten Farbe ausmalt. Das Programm orientiert sich dabei an den Grenzlinien. Beim Erreichen einer Linie läuft der Cursor in einer anderen Richtung weiter, bis er wieder an eine Umrißlinie stößt. Für einen korrekten Programmablauf müssen die Grenzlinien genau definiert und vollständig sein. Auch eine kleine Unterbrechung führt unvermeidlich zum „Auslaufen" der Farbe.

Fehlererkennung

Die BASIC-Dialekte der meisten Heimcomputer unterstützen die Fehlersuche durch differenzierte, eindeutige Fehlermeldungen und die Möglichkeit, ein abgestürztes Programm nach der Eingabe anderer Variablenwerte neu zu starten. Viele Dialekte verarbeiten auch Befehle wie ON ERROR GOTO: Der Programmablauf wird zu einer Routine für die Fehlerbehandlung „umgeleitet". Dafür fügt man beispielsweise diese Zeile in das Programm ein:

```
30 ON ERROR GOTO 20000:REM
    ROUTINEN ZUR FEHLERBEHANDLUNG
```

Tritt ein Fehler auf, würde das Programm auf Zeile 20000 verzweigen. ON ERROR setzt im allgemeinen auch noch zwei Variablen, eine für den Fehlertyp, die zweite für die Zeilennummer, in der der Fehler aufgetreten ist. Die Variablennamen wie auch die Werte des Fehlertyps sind je nach verwendetem Rechner unterschiedlich – genauere Angaben können Sie dem Handbuch Ihres Rechners entnehmen.

Gut durchdachte Programme kommen mit einer einzigen ON-ERROR-Routine aus. Natürlich kann eine solche Routine keine Syntaxfehler beheben oder das Überlaufen eines Stack-Registers verhindern. Der Nutzen liegt eher in einem sauberen Programmabschluß ohne offene Files. Außerdem erfährt der Anwender ungefähr, was den Fehler hervorgerufen hat.

Manche Fehler, etwa eine Division durch die Null, könnten zwar mit einer ON-ERROR-Routine behandelt werden, aber man sollte trotzdem darauf verzichten. Dafür sprechen mehrere Gründe:

- Der ON-ERROR-Befehl mit einem Sprung ins Hauptprogramm beinhaltet je einen Ein- und Ausgang – eine Verletzung der Regel für strukturierte Programmierung, die nur einen Ein- und Ausgang pro Teilprogramm vorsieht.
- Eine Division durch Null sollte dort verhindert werden, wo dividiert wird: Ein guter Programmierer vermeidet Algorithmen, die zu einem Systemabsturz führen könnten. Verlangsamen die dafür nötigen Vorkehrungen das Programm übermäßig, sollte man es entsprechend umschreiben.
- Fehlerbehandlungs-Routinen können schnell zu unhandlichen Ketten von IF...THEN...ELSE-Befehlen mit diversen Verzweigungen anwachsen. Sie sind mit den Zeilennummern des Restprogramms verbunden und müssen daher bei jeder Änderung eines Unterprogramms neu geschrieben werden. Zudem sind sie schwierig zu planen und noch schwieriger zu prüfen. Mängel bei der Fehlerbehandlung können den Programmablauf in völlig unvorhergesehener Weise stören.

Der Faktor Mensch

Ein wichtiger Aspekt beim Programmaufbau ist die „Schnittstelle Mensch-Maschine", also der Teil des Programms, das die Datenübertragung vom Anwender zur Maschine und von der Maschine zum Anwender steuert. In diesem Artikel untersuchen wir die Faktoren, die dabei berücksichtigt werden müssen.

Viele Jahre lang war die Computerprogrammierung ein beinahe „mystisches" Gebiet, das nur von Profis verstanden wurde. Vor dem Auftauchen des Microcomputers mit seiner Schreibmaschinentastatur wurden Programme oft byteweise über Schalter an der Vorderseite der Computer, per Lochstreifen oder über Fernschreiberkonsolen eingegeben.

Im Vergleich dazu wird der Anwender heute regelrecht verwöhnt. Die Computerbesitzer brauchen sich nicht mehr mit dem Maschinencode abzugeben, und der Begriff „benutzerfreundlich" besagt, daß sich die Micros unabhängig von Vorkenntnissen einsetzen und programmieren lassen. 1982 veröffentlichte das Alvey-Komitee in dem Bericht „Ein Programm für die fortgeschrittene Informationstechnologie", daß die Schnittstelle Mensch-Maschine als einer der vier Haupt-Forschungsbereiche angesehen werde. Die anderen drei Gebiete sind die Konstruktion von Software, integrierte VLSI-Schaltungen (Very Large Scale Integration) und Systeme zur Wissensverwaltung.

In fast jeder Anwendung spielt die Wechselwirkung bzw. der Datenaustausch zwischen Computer und Anwender eine wesentliche Rolle. Dieser „Dialog" wird über die Ein- und Ausgabekomponenten des Computers (E/A) geleitet, wobei die Tastatur die wichtigste Eingabequelle darstellt und der Bildschirm das wichtigste Ausgabeinstrument. Eingaben sind aber auch über Joysticks, Mäuse, Touch-Screens und andere Geräte möglich, während sich für die Ausgabe auch Drucker, Tongeneratoren (Sprache) und sogar Roboter einsetzen lassen.

Doch außer den eingesetzten E/A-Geräten bestimmt hauptsächlich die Software den Dialog zwischen Mensch und Computer. So steuert beispielsweise das Betriebssystem viele Funktionen von Tastatur und Bildschirm. Es bestimmt die Wiederholgeschwindigkeit einer gedrückten Taste, den Zeitraum zwischen den einzelnen Wiederholungen und bewahrt auch die Zeichen auf, die schneller eingegeben wurden, als sie der Bildschirm anzeigen kann. Einige Betriebssysteme können nur ein Zeichen zwischenspeichern, die meisten Heimgeräte haben aber Buffer für zehn oder mehr Zeichen.

Zeichenbuffer können allerdings auch Probleme verursachen. Erfahrene Anwender, die mit menügesteuerten Systemen arbeiten, wissen beispielsweise, daß sie aus dem Hauptmenü die 2 wählen müssen, aus dem nächsten Menü 5, dann 3,4,6 etc. Sie sind mit dem System vertraut und geben diese Ziffern mit hoher Geschwindigkeit ein. Bei einem Buffer für zehn Zeichen entstehen keine Probleme, da der Computer sich alle Tasten in der richtigen Reihenfolge „merkt". Mit einem Ein-Zeichen-Buffer kann die Darstellung des zweiten Menüs jedoch mehr Zeit in Anspruch nehmen, als die gesamte Eingabe dauert. Da nur das letzte Zeichen im Buffer bleibt, wird statt der 5 und der 3 etc. nur die Eingabe 6 ausgeführt (als letztes vom Buffer gespeichert), und das Programm bricht ab.

Aber auch große Buffer sind nicht unproblematisch. Bei Menüprogrammen, die erst nach einiger Zeit auf Eingaben reagieren, kann der Anwender den Eindruck erhalten, daß überhaupt nichts passiert. Er versucht die Eingabe

nochmals und drückt so lange auf die Tasten, bis etwas passiert. Wenn das Programm dann versucht, die im Buffer gespeicherten Anweisungen auszuführen, können unerwartete Ergebnisse entstehen.

Das „Räumen" des Arbeitsspeichers von nicht mehr benötigten Variablen stellt eine weitere Problemquelle dar. Da das Programm bei dieser Arbeit lange Zeit zu „hängen" scheint, sind viele Anwender versucht, diesen Zustand irgendwie zu beenden. Besonders große Pro-

gramme mit zahlreichen String-Variablen können für die Speicherbereinigung viel Zeit benötigen. In einigen BASIC-Versionen läßt sich in festen Zeitabständen eine Bereinigung durchführen. Dabei sollte eine Meldung anzeigen, daß der Computer beschäftigt ist.

Auch die Art, wie Programmiersprachen Ein- und Ausgaben handhaben, beeinflußt die Schnittstelle Mensch-Maschine. BASIC-Versionen mit eingebauter Cursorsteuerung und Grafikbefehlen gestalten Bildschirmeingaben einfacher. Überhaupt ist BASIC gut mit Ein- und Ausgabebefehlen ausgerüstet. So genügen für einfache Programme oft INPUT und PRINT, während die echte Eingabesteuerung (die beispielsweise auch Eingabefehler abfangen soll) mit GET$, INKEY$, INPUT$() und ähnlichen Befehlen programmiert werden kann. Weiterhin ist PRINT USING ein außerordentlich flexibler Befehl für die Ausgabeformatierung. Er kann Dezimalpunkte genau untereinandersetzen und Textspalten justieren.

Der Anwender ist unberechenbar

In jedem System Mensch-Maschine ist der Anwender das am wenigsten berechenbare Element. Wie die anderen Komponenten folgt auch seine Arbeit jedoch gewissen Mustern, die bei der Konstruktion der „Schnittstelle" daher unbedingt berücksichtigt werden müssen.

Micronet 800

Eine der grundlegenden Eigenschaften von Mensch und Computer ist die Fähigkeit, „Informationen verarbeiten zu können". Bei Menschen ist jedoch die Informationsmenge, die sie in ihren „Arbeitsspeicher" aufnehmen können, begrenzt. Man geht davon aus, daß von den meisten Informationsarten etwa sieben verschiedenen „Daten" gleichzeitig im Gedächtnis gespeichert werden. Die Menge der erinnerten Informationen hängt oft von der Struktur und Bedeutung dieser Merkmale ab. Sind die Informationen aus zufällig auftretenden Zeichen zusammengesetzt, dann besteht jede Erinnerung nur aus einzelnen Zeichen. Sind sie jedoch nicht zufällig, sondern bilden beispielsweise bekannte Nachnamen, dann kann jedes „Datum" durchaus einen ganzen Namen enthalten. Eine Verbesserung der Informationsstruktur kann dabei die Erinnerungsfähigkeit des Anwenders vergrößern.

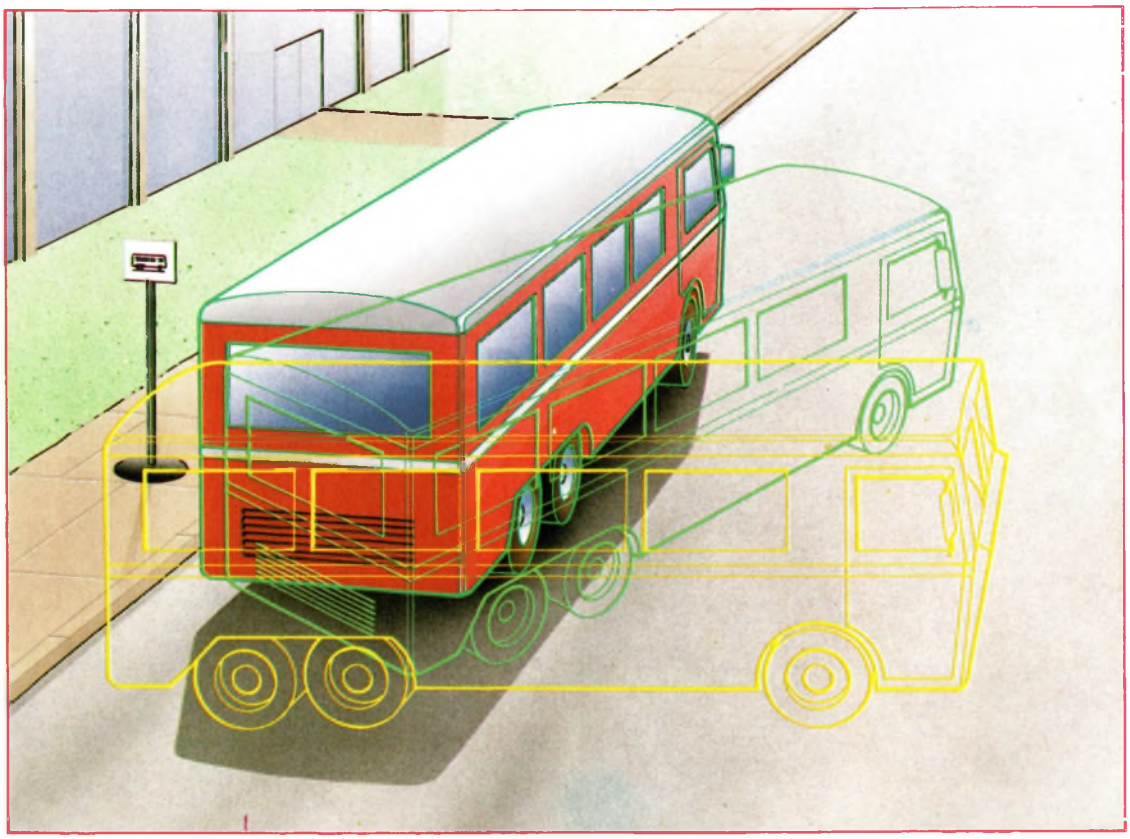
Der Mensch kann auf mehrere Arten unterstützt werden, Informationen für den Einsatz eines Computers zu strukturieren. Eine dieser Methoden stellt eine Beziehung zwischen Daten und vertrauten und verstandenen Strukturen her. Der „Schreibtisch" des Apple Lisa arbeitet nach diesem Schema, und auch Kalkulationssysteme werden oft wie Bücher mit Seiten, Indizes etc. angelegt. Eine andere Methode macht den Anwender mit neuen Strukturen vertraut: Mit wiederholten Beispielen und ausführlichen Erklärungen bringen dabei die Programme selbst dem Anwender bei, wie er seine Informationen strukturieren soll. Dieses Training erfordert aber viel Zeit und Aufwand. Ausführliche Informationen, „Hilfsmenüs" und „Merker" können zwar eine direkte Hilfe bieten, sind aber nicht immer leicht zu bedienen.

Schließlich fördert schon die Anordnung von Informationen das Verständnis. Farbliche Gestaltung und ein spezieller Bildschirmaufbau helfen beim Auffinden der gewünschten Daten. Beispiele dafür sind die Farbcodes von Bildschirmtext, Prestel etc. Auf einer typischen Seite sind die „Überschrift" und die „Fußzeile" in Blöcken gleicher Farbe gehalten. Vor einer einheitlichen Hintergrundfarbe erscheint der Text in zwei weiteren Farben, die von Absatz zu Absatz wechseln. Wichtige Wörter werden in einer anderen Farbe markiert. Dieser Aufbau gibt dem Anwender die Möglichkeit, nur die gewünschte Information zu lesen und Bereiche zu ignorieren, die für ihn keinen unmittelbaren Wert haben. Zu viele Farben können aber auch Verwirrung stiften. Eine gute Faustregel ist daher: Man sollte niemals mehr als vier Farben gleichzeitig bei der Gestaltung des Bildschirms einsetzen.

Apple Macintosh

Die drei Fotografien zeigen Betriebssysteme von Microcomputern, die einen unterschiedlichen Grad an Benutzerfreundlichkeit haben. Auf dem ersten Bild versucht ein Anfänger, mit dem CP/M-Betriebssystem klarzukommen. Da CP/M keine eingebauten Hilfen besitzt, ist eine genaue Kenntnis der Befehle nötig, bevor das System richtig eingesetzt werden kann. Das zweite Beispiel zeigt ein Menüsystem – das „Log-on"-Menü für Micronet 800 auf dem Acorn B –, dessen Optionen mit Nummern versehen sind. Der Anwender braucht die gewünschte Funktion nur mit der entsprechenden Zahl aufzurufen. Da auch hier der Bildschirm nicht allzuviele Informationen bietet, muß der Anwender wissen, welche Funktionen sich hinter den Bearbeitungsmöglichkeiten verbergen. Unser letztes Bild zeigt das Betriebssystem des Apple Macintosh, das sichtbare Hilfen, grafische Darstellungen und einfache, leicht zu verstehende Menüs enthält. Mit der Maus wird jeweils eines der Piktogramme angesteuert und durch Knopfdruck gewählt.

Höhere Einsichten



Bei einigen Mustererkennungs-Systemen bedient man sich des „Top-Down"-Vergleichs, in dem ein Muster oder eine Szene auf einen bestimmten Gegenstand hin abgetastet wird. Eine Rahmendarstellung des Gegenstandes wird in verschiedenen Winkeln auf den Bildschirm projiziert, bis eine Projektion gefunden wird, die auf den Umriß des gesuchten, tatsächlichen Gegenstandes paßt.

Da Sehen und Verstehen zwei eng zusammenhängende Faktoren sind, die zum Begreifen unserer Umwelt beitragen, ist es oft schwer, die beiden eindeutig voneinander zu trennen.

Schönheit entsteht im Auge des Betrachters", heißt ein Sprichwort. Genauer aber muß es heißen, daß Schönheit im Gehirn des Wahrnehmenden entsteht oder – noch genauer – in einer komplexen Kette nervlicher Prozesse, die irgendwo auf der Netzhaut beginnt und in der Großhirnrinde am Hinterkopf endet.

Die visuelle Wahrnehmung ist so untrennbar mit dem Verstehen unserer Umwelt verbunden, daß wir häufig sagen „ich seh' schon" oder „ich sehe ein, daß", wenn wir meinen „ich verstehe". Irgendwie muß auch der Computer die über eine Kamera oder ein anderes lichtempfindliches Gerät erhaltenen Informationen umsetzen, um zu erfahren, was damit über den Zustand seiner Umgebung ausgesagt werden kann. Die Beschaffung der Information ist leicht – das Problem ist ihre Interpretation.

Der Prozeß der Umwandlung von Informationen in Aussagen ist vom Zeitablauf her in drei Schritte zu untergliedern:

1. Bild-Entwicklung: Die Umwandlung eines verwischten oder unscharfen Bildes in ein scharfes (relativ leicht lösbar).
2. Mustererkennung: An- oder Abwesenheit eines bestimmten Gegenstands feststellen (schwieriger).
3. Bildverständnis: Erarbeiten und verstehen dessen, was in der realen Welt geschieht (sehr schwierig).

Wenngleich der dritte Schritt – echtes Verstehen des Computers – noch nicht erlangt wurde, hat man auf den beiden vorhergehenden Ebenen bereits gute Ergebnisse erreicht.

Mustererkennung ist eine herausfordernde Aufgabe. Bei dieser Methode werden Bilder klassifiziert, indem diese mit einer begrenzten Reihe von Alternativen verglichen werden, so etwa bei optischen Zeichenerkennungs-Systemen mit den Buchstaben des Alphabets. Typisch ist dabei, daß gewisse Merkmale des digitalisierten Bildes isoliert untersucht werden. Das System kann, entsprechend dem Vorhandensein oder Nichtvorhandensein dieser Merkmale, die Muster erkennen und klassifizieren.

Die Erkennung mancher Merkmale, wie beispielsweise diagonaler Striche, scheint uns

klar und einfach. Oft aber erfolgt die Erkennung durch optische Systeme aus rein willkürlichen Berechnungen, die aufgrund der Bilddaten durchgeführt werden und somit unzureichende Ergebnisse hervorbringen.

Historisch gesehen gibt es zwei voneinander abweichende Wege zur „Computervision" im Sinne von Bilderkennung – die „Top-Down"- oder Muster-Methode und die „Bottom-Up"- oder auf Daten basierende Methode.

Die auf der Muster-Methode basierende Szenenanalyse wird zuweilen als „kontrollierte Halluzination" bezeichnet. Das System verfügt über ein internes Muster dessen, was es betrachtet (zum Beispiel einen Doppeldecker-Bus), und es projiziert das Modell in verschiedenen Richtungen auf die Bildfläche. Dann sucht es nach Übereinstimmungen zwischen dem „eingebildeten" Muster und dem, was tatsächlich zu sehen ist.

Verstehen durch Abstraktion

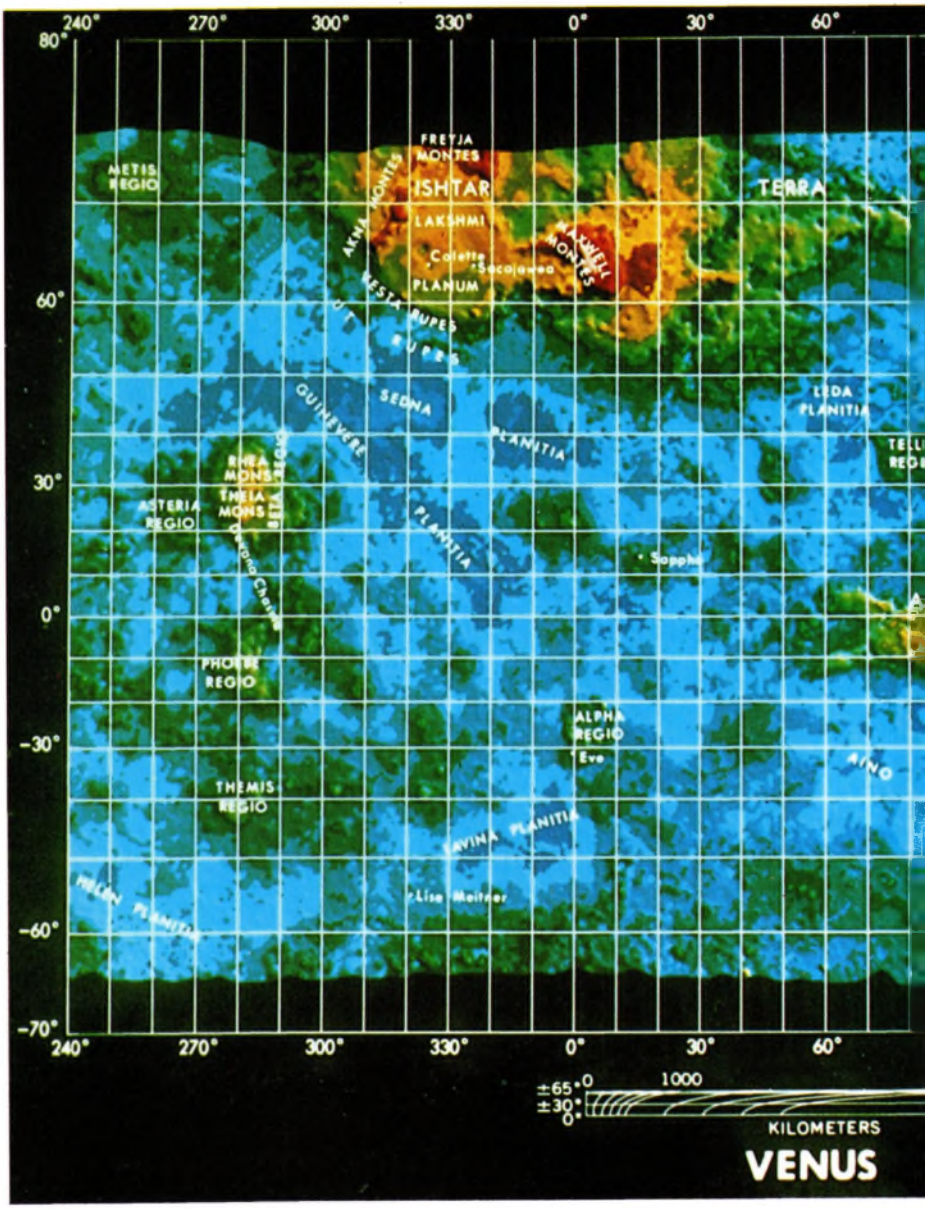
„Bottom-Up"-Systeme tasten die Bilddaten nach Linien, Kanten und anderen markanten Merkmalen ab. Damit wird eine einfache Beschreibung des Bildes konstruiert, das man „Rohskizze" nennt. Es handelt sich dabei um eine sehr abstrakte Darstellung. Der Sinn ist, daß man durch Verzicht auf Details die Skizze leichter mit gespeicherten Mustern vergleichen kann.

In der Praxis finden beide Erkennungsmethoden Anwendung. Erst seit kurzem gibt es in diesem Bereich auch lernfähige Systeme. Sie basieren auf vorprogrammierter Intelligenz.

Eine bemerkenswerte Ausnahme ist Igor Aleksanders WISARD (Wilkie, Aleksander & Stonham's Recognition Device). Dieses System kann selbst den vergleichsweise feinen Unterschied zwischen einem lächelnden und einem ernsten menschlichen Gesicht erkennen, und es kann sein Wissen sogar auf Gesichter übertragen, die es nie zuvor gesehen hat. WISARD arbeitet in Echtzeit (25 Bilder pro Sekunde).

WISARD arbeitet durch Zuweisung von jeweils acht Punkten zu einem ausgewählten RAM-Speicherbereich. Diese Achtergruppen (octuple) dienen als Merkmal-Detektoren, die sich jeweils in einem von 256 Zuständen befinden können (0 bis 255), – abhängig vom Status der jeweiligen acht Punkte. In der Lernphase wird eine Eins an der jeweiligen Adresse im RAM gespeichert, wenn ein bestimmtes Bild vorhanden ist. In der Erkennungsphase ist die Eins an der bestimmten Adresse Beweis, daß das „gedachte" Bild erneut vorhanden ist.

Durch Verwendung einer großen Zahl von Achtergruppen oder sogenannten „Unterscheidern" (Diskriminatoren) wird das System unempfindlich gegenüber falschen Daten (Störbildern). In Aleksanders System werden ein 512 × 512 messendes Bildraster und über 32 000 Achtergruppen verwendet.



0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	1	1	0	0
0	1	1	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	0	0	0	1	0

0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	1	1	1	0	0	0
0	1	1	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	0	1	0

Der Unterschied zwischen R und A

In der realen Welt gibt es nur selten exakt definierte Daten. Das obige Beispiel zeigt digitalisierte Formen der Buchstaben A und R. Doch keines der Muster ist genau definiert, sondern hat Übergänge und Abweichungen. „Störbilder" oder „Stördaten" wie diese verursachen Probleme bei der Mustererkennung. Wissen Sie, was welcher Buchstabe ist?

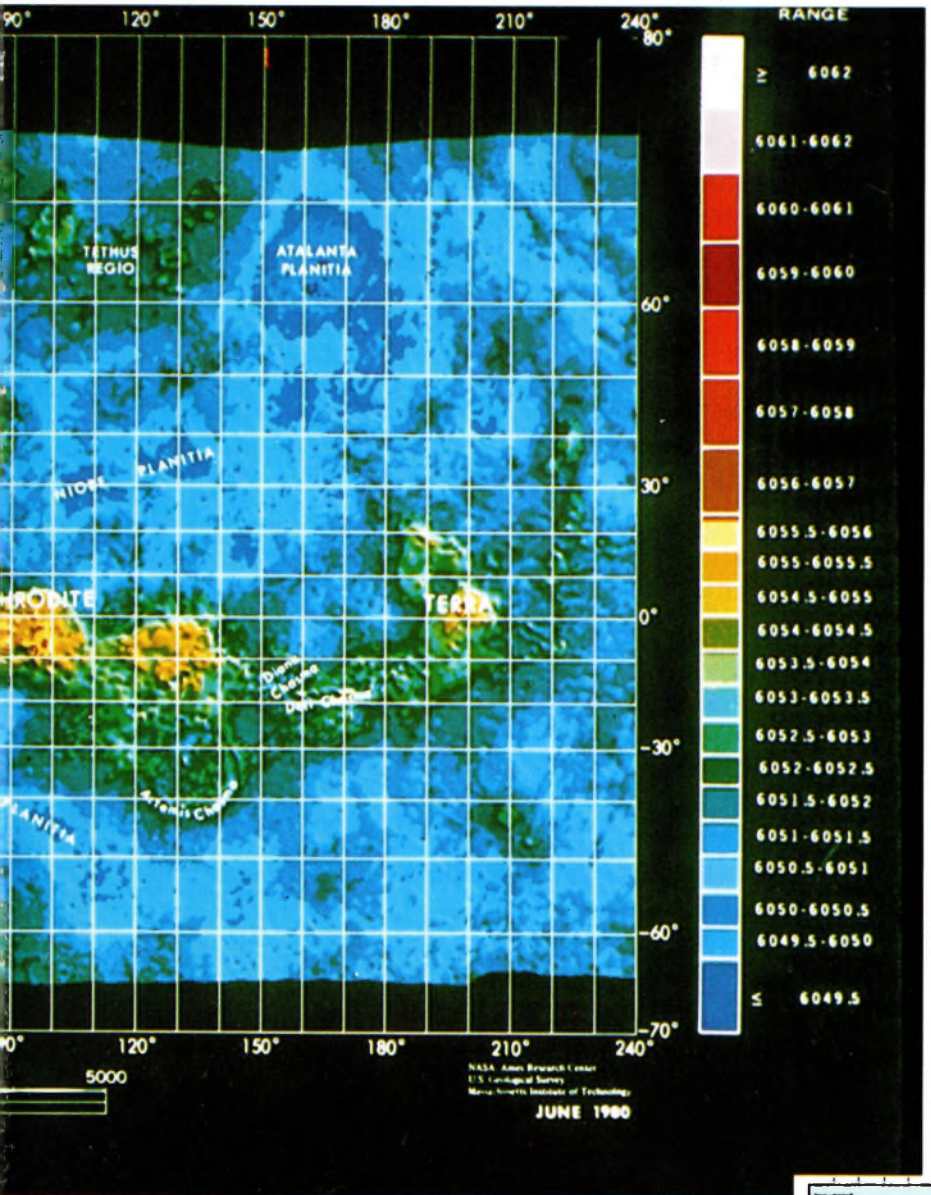


Bild-Verarbeitung

Das von einer Raumsonde aufgenommene Foto wurde von einem Computer bearbeitet, um die unterschiedlichen Oberflächenstrukturen zu verdeutlichen. Dazu verwendete man „Falsch-Farben". Interessant an diesem Bild des Planeten Venus ist, daß alle Farben „falsch" = künstlich erzeugt sind. Ohne Computerhilfe wäre ein solches Bild nicht sichtbar, da die Venus stets von dicken Wolken umgeben ist.

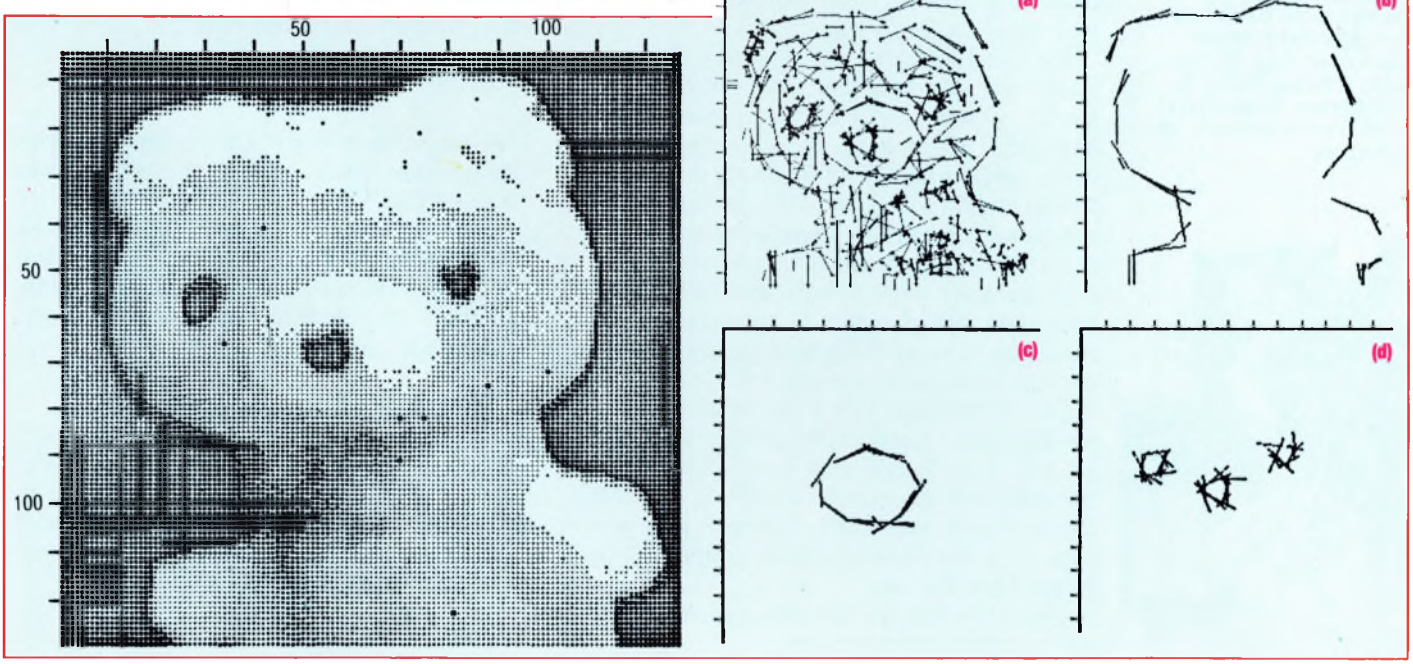
In der Astronomie haben sich solche Bilder als sehr nützlich erwiesen. Man nutzt dabei die Vorteile von Radio-, Infrarot- und Ultraviolett-Wellen ebenso wie das sichtbare Licht.

Auf der Erde beobachtet man die Meere mit Sonar-Wellen, um akustische Bilder zu erzeugen. Hierbei findet das Echolot-Prinzip Anwendung.

Andere Arten der „Bild-Verarbeitung" werden genutzt, um verzerrte oder unscharfe Bilder „zu reinigen".

Rohskizzen

Es gibt eine Methode der Bildverarbeitung, die nicht auf vorhandenen Erkenntnissen und gegebenem Wissen basiert. Die Rohskizze (a) isoliert einfache Konturen aus dem Abbild des Teddybären, indem es den Kontrast benachbarter Regionen vergleicht. Weitere Auszüge aus den Rohskizzen (b) bis (d) stellen wichtige Gruppen dar, die zur Erkennung des Originalbilds beitragen.



Routine-Prüfung

Nachdem in unserer Artikelserie über die Technik des Programmierens bisher Anregungen zum Programmentwurf und zu den Details der Ausführung im Vordergrund standen, geht es hier abschließend um das Testen des fertigen Programms.

Ein Testdatensatz einschließlich der handgerechneten Ergebnisse für das Beispiel aus den Abbildungen 2 und 3 könnte folgendermaßen aussehen:

	INPUT		OUTPUT
LEVEL	HITS	BONUS	BONUS
6	10	200	1300
4	10	550	2300
7	10	550	3950
4	10	200	800
7	10	200	1400
1	20	2500	2600
1	20	550	550
6	5	200	300
6	50	200	300
4	5	2500	2600
7	50	2500	2600
4	50	550	550
7	5	550	550

Ein Hauptvorteil von Interpretersprachen wie BASIC besteht darin, daß schon während des Programmeditierens ein Funktionstest möglich ist – man kann jederzeit RUN eintippen und verfolgen, was daraufhin passiert. Bei den meisten Rechnern können Sie den Probelauf auch mit BREAK unterbrechen, mit PRINT die aktuellen Werte von Schlüsselvariablen ausgeben lassen, diese verändern und das Programm dann mit CONTinue fortsetzen. So lassen sich formale und teilweise auch logische Fehler weitgehend, schon während der Programmeingabe, finden und korrigieren. Ein Test des fertigen Programms in seiner endgültigen Form bleibt aber trotzdem unerläßlich.

Diese letzte Prüfung soll sicherstellen, daß das Programm genauso arbeitet wie vorgesehen. Für jeden zulässigen Eingangsdatensatz muß das richtige Ergebnis herauskommen, und für jede unzulässige Eingabe die entsprechende Fehlermeldung. Selbst wenn Ihr Programm nur zwei Integer-Zahlen addiert und die Summe ausdruckt, müßten Sie etliche Milliarden von Wertepaaren durchspielen.

Testmethoden

Sie könnten auch daran denken, jeden möglichen Weg durch das Flußdiagramm vom Anfangs- zum Endpunkt zu prüfen. Jede neue Verzweigung eröffnet einen Alternativweg, und jeder neue Schleifendurchlauf wieder zusätzliche Kombinationen. In Abbildung 1 wird eine einfache Schleife mit Bedingungsabfragen (IF...THEN) für vier interne Wege dargestellt, die zehnmal durchlaufen wird. Dabei gibt es $4^{10}=1\,048\,576$ verschiedene Wegvarianten vom Start bis zum Ende – verblüffend für ein Programm, das sich in wenigen Zeilen niederschreiben läßt. Also scheidet auch das Testen der Wegvarianten aus. Welche Möglichkeit bleibt dann noch?

Überraschenderweise gar keine – Sie können ein komplexes Programm in realistischer Zeit nicht im genannten Sinn erschöpfend prüfen. Deshalb verläßt man sich beim Testen auf das Gesetz von der abnehmenden Effizienz jeder Fehlersuche: Die Anzahl der Fehler, die mit jedem neuen Durchgang gefunden werden, nimmt kontinuierlich ab. Man hört auf, wenn die weitere Fehlersuche kostspieliger würde als die Schadenbeseitigung bei eventuell verbliebenen Restfehlern.

Unabhängig davon lohnt es sich, über rationelle Testmethoden nachzudenken. Eine vernünftige Annahme ist zum Beispiel, daß ein Programm, wenn es einen bestimmten Satz von Eingangsdaten richtig verarbeitet, auch mit gleichgearteten neuen Zahlenwerten korrekt verfahren wird. Wenn ein Unterprogramm mit einer positiven Integer aus seinem Definitionsbereich einwandfrei läuft, sollte es auch mit jeder anderen zugelassenen positiven Integer funktionieren. Das führt zum „Äquivalenzklassen"-Verfahren: Sie denken sich Testdatensätze aus, die jeweils für eine ganze Klasse von gleichartigen Werten repräsentativ sind. Wird etwa irgendwo abgefragt, ob ein Parameter X zwischen 1 und 100 liegt, dann braucht man drei Testgrößen X_1, X_2 und X_3 mit den Werten $X_1<1$, $X_2>100$ und $1\leq X_3\leq 100$.

Vierergespann

Selbst ein relativ einfaches Gebilde wie diese Schleife mit vier internen Parallelpfaden läßt sich nicht erschöpfend austesten, weil es zu viele Durchlauf-Varianten gibt: Bei zehn Zyklen sind über eine Million unterschiedliche Streckenzusammenstellungen möglich.

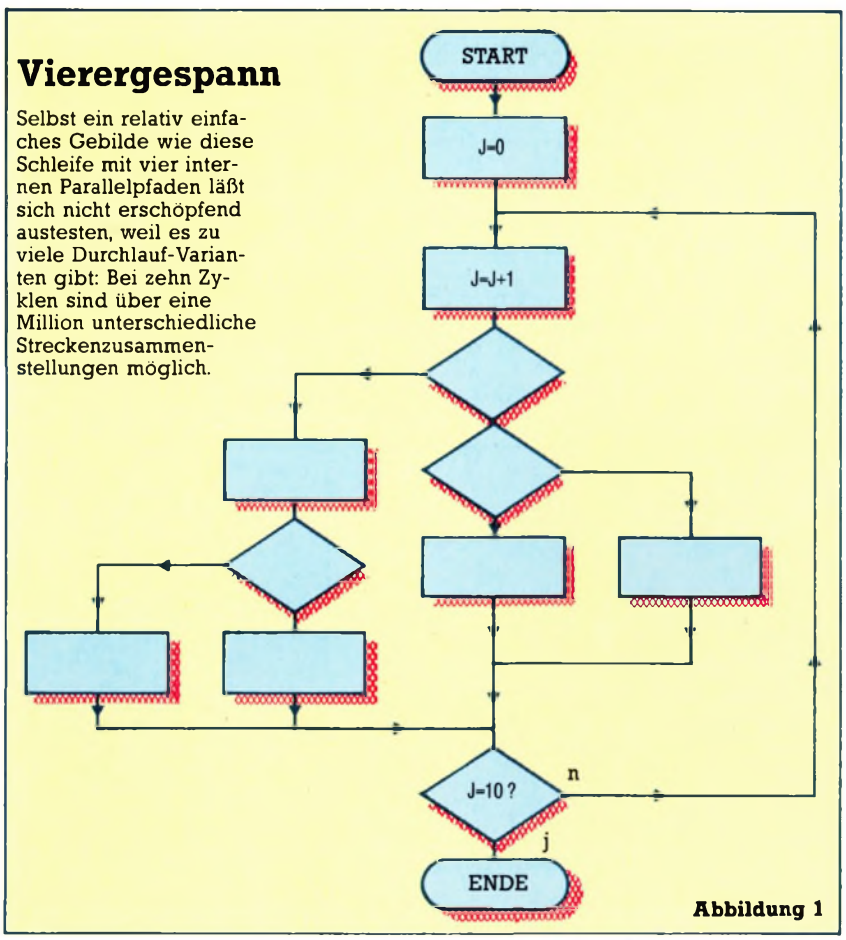


Abbildung 1

Die Überprüfung der Programmwege läßt sich so vereinfachen, daß auf die Einstiegsadresse jeder Routine verzweigt wird und die Testdaten so gewählt werden, daß jeder interne Ast einmal durchlaufen wird. In Abbildung 2 ermittelt ein Programm für ein Computerspiel aus der alten Punktzahl BONUS, dem Schwierigkeitsgrad LEVEL und der Trefferanzahl HITS den neuen BONUS. Der Text dazu könnte lauten:

```
6030 IF LEVEL>2 AND HITS=10 THEN
        BONUS = BONUS*LEVEL
6040 IF LEVEL = 6 OR BONUS> 2000 THEN
        BONUS = BONUS + 100
```

Um die möglichen Ergebnisse jeder Bedingungsabfrage abzudecken, ist zu überlegen, welche Eingangswerte die Erfüllung bzw. Nichterfüllung der Bedingungen bedeuten. Die Abfragen in Abbildung 2 betreffen je zwei durch AND/OR verknüpfte Variablen. Entscheidend ist, wie sich die Kombination der Variablen auswirkt. Wenn man zum Test der ersten Abfrage beispielsweise für LEVEL einmal 4 und einmal 1 und für HITS jedesmal 10, 5 und 20 vorsähe, würden mit LEVEL=4 zwar die drei Möglichkeiten für HITS ausprobiert, mit LEVEL=1 aber nicht. Hier ist ein Teil der Entscheidung durch einen anderen „maskiert".

Solche Abfragen sollten Sie zum Testen logisch zerlegen, wie es in Abbildung 3 dargestellt wird. Die vier einfachen Entscheidungen führen hier oben und unten auf je drei verschiedene Wege, je nach den Eingangswerten. Nützlich ist eine Zusammenstellung der Ja/Nein-Bedingungen für jede Abfrage:

Abfrage	1	2	3	4
Ja	LEVEL>2	HITS=10	LEVEL=6	BONUS>20000
Nein	LEVEL=2 LEVEL<2	HITS<10 HITS>10	LEVEL<6 LEVEL>6	BONUS=2000 BONUS<2000

Damit lassen sich repräsentative Testdaten herleiten. Zum Beispiel muß für die Route a/d/f/i der LEVEL>2, nämlich =6 sein, die HITS müssen <>10 sein und der (alte) BONUS ist beliebig, weil er nicht abgefragt wird. LEVEL=6, HITS=20 und BONUS=150 wäre ein möglicher Wertesatz (natürlich nicht der einzige). Der Pfad a/b/e/h/j würde mit LEVEL=4, HITS=10 und BONUS=600 (vor Multiplikation mit LEVEL!) angesteuert.

Unerläßlich ist eine Handrechnung mit jedem Datensatz als Gegenprobe zum Computer. In der Tabelle links wird der ganze Testdatensatz durchgerechnet.

Wie kann man nun mit ähnlichen Prüfmethoden großen Programmen zuleibe rücken, ohne von ihrer Komplexität erdrückt zu werden? Erste Voraussetzung ist eine übersichtliche Programmstruktur. Ist das Programm nach dem Prinzip der schrittweisen Verfeinerung aus unabhängigen Modulen aufgebaut, können Sie alle einzeln von oben nach unten testen (siehe Abbildung 4).

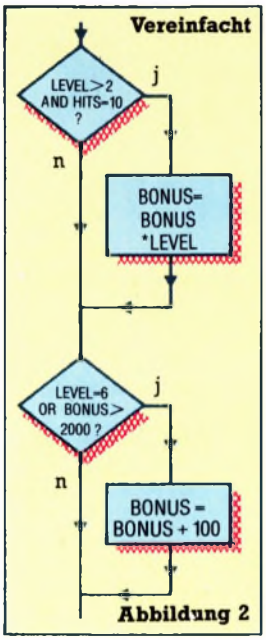


Abbildung 2

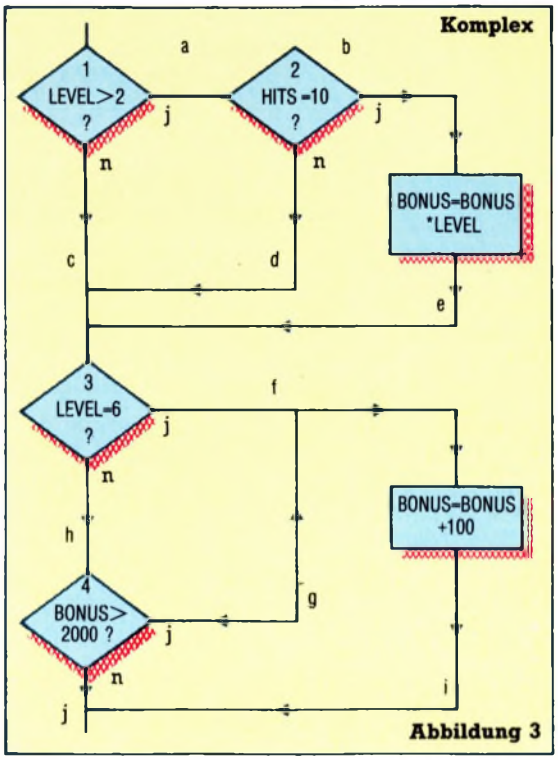


Abbildung 3

Am besten zerlegen Sie komplexe Entscheidungen und beschriften die Flußlinien, um die Tests zu vereinfachen.

Bei diesem Vorgehen kommt jedes Segment erst an die Reihe, wenn die darüberliegenden „Treibermodule" ausgetestet sind, so ist auch die Parameterversorgung gesichert. Die tiefer stehenden, ungetesteten und daher unzuverlässigen Module werden zunächst durch Scheinroutinen („Dummies") ersetzt – ein paar Programmzeilen, die bei Aufruf vorher eingespeicherte Testdaten abliefern. Das Ganze hat den Charakter eines „Testrahmens" (Test Harness), in den die Segmente zur Erprobung eingespannt werden. In Abbildung 4 sind die Module 1, 2 und 3 bereits ausgetestet, während 5, 6 und 7 durch Dummies simuliert werden. Die Nummer 4 wird gerade untersucht.

Wenn Sie die Testdatensätze und Ergebnisse aufheben, muß bei später auftretenden Fehlern nicht alles wiederholt werden, und Sie können anhand der Dokumentation die erforderlichen Ergänzungstests festlegen.

Wenn Ihr Programm eine „umgekehrte Baumstruktur" (Top-Down) aufweist, also vorschriftsmäßig nach dem Prinzip der schrittweisen Verfeinerung aufgebaut ist, können Sie jedes Segment unmittelbar nach Fertigstellung sowohl isoliert wie im Zusammenspiel mit anderen prüfen.

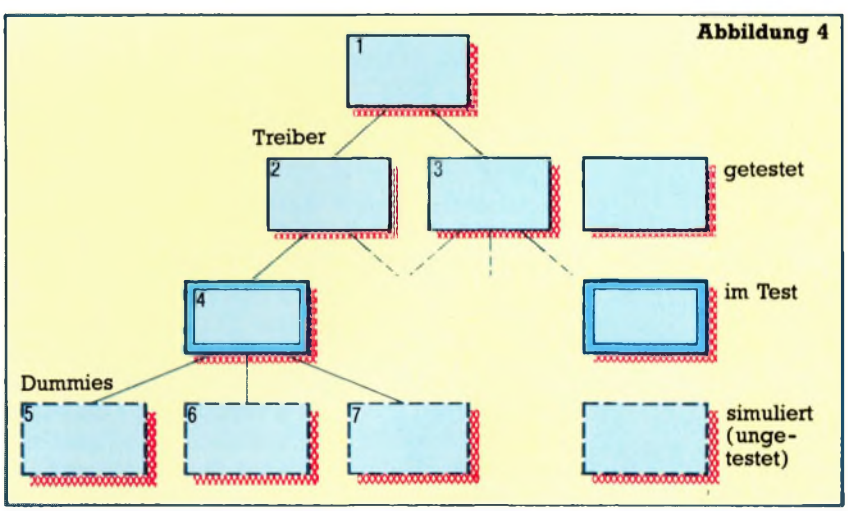


Abbildung 4

Grafikrahmen

Eins der größten Probleme der WIMP-Umgebungen ist die Übertragbarkeit der Programme auf andere Computer. Ein Grafikstandard für Kernroutinen bietet eine Lösung.

Die Erstellung einer WIMP-Umgebung für einen bestimmten Computertyp ist teuer. Überdies lassen sich diese speziell entwickelten Grafikanwendungen nicht auf andere Geräte übertragen. Übertragbarkeit würde bedeuten, daß das normale Betriebssystem mit einem „Grafikrahmen" arbeitet, der Fenster und Zeiger steuert und auch Programme ablaufen läßt. Nach Beendigung eines Vorgangs wird die Steuerung dabei nicht an das normale Betriebssystem zurückgegeben, sondern an den WIMP-Rahmen, der die Routinen des eigentlichen Betriebssystems (für Dateiverwaltung etc.) anspricht.

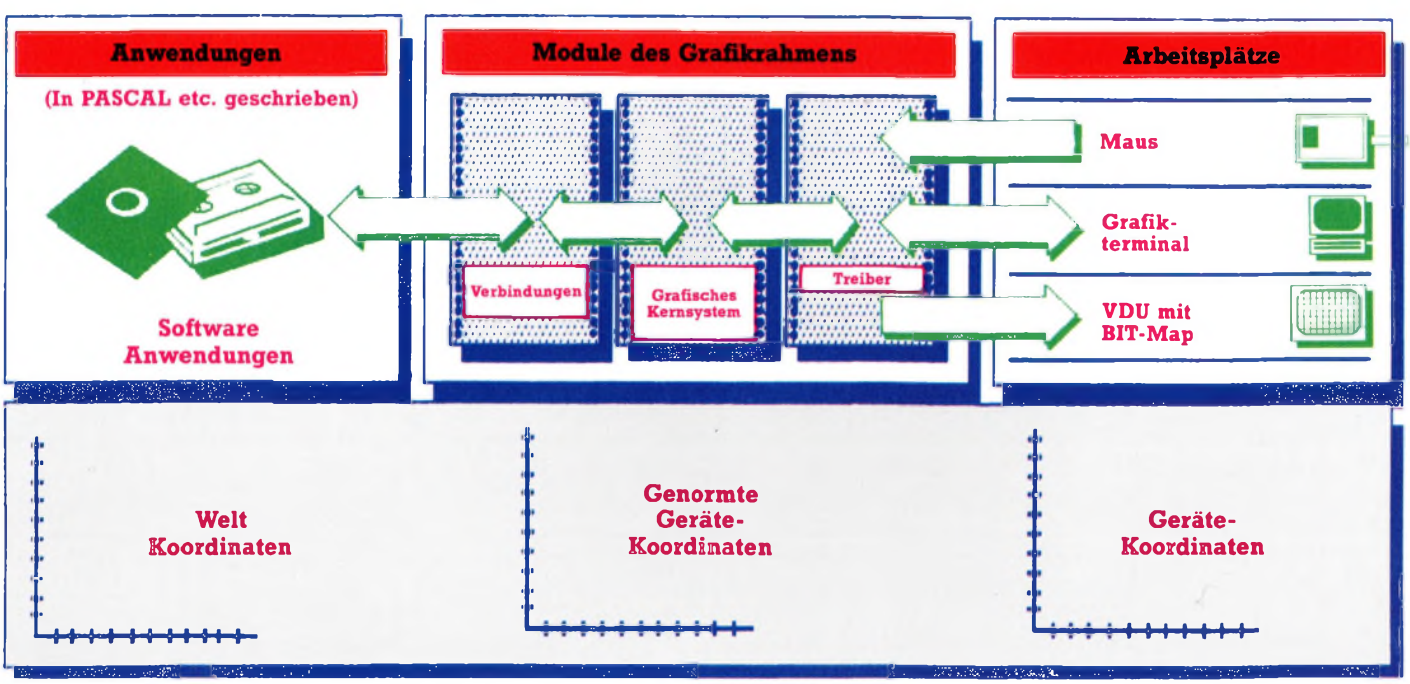
Die große Vielfalt der Grafiksoftware spricht zwar gegen eine Standardisierung dieser Umgebung, doch gibt es inzwischen Rahmenprogramme, die auf einem breiten Hardwarespektrum laufen, darunter TopView von IBM, MS-Window von Microsoft und GEM.

1977 wurden erstmals Bedingungen für ein „Grafisches Kern System" (GKS) veröffentlicht, das die Grafikprogrammierung systemunabhängig machen sollte. Inzwischen bieten mehrere Hersteller GKS an, doch blieb der Anwenderkreis wegen des hohen Preises auf CAD/CAM-Anwendungen und Universitäten beschränkt. Digital Research war die erste auf Microcomputer spezialisierte Firma, die für diesen Bereich das Rahmenprogramm GSX (Grafics System Extension) herausbrachte. GSX baut auf GKS auf, ist nicht kompatibel.

GSX ist eine Betriebssystemerweiterung der von DR entwickelten CP/M-86-Version und muß für jede Maschine angepaßt werden. Ist GSX erst einmal installiert, können Programme darauf zugreifen, ohne sich darum kümmern zu müssen, wie Hard-, Soft- oder Firmware diese Funktionen ausführen. GSX steuert die Hardware wie ein Betriebssystem, besitzt aber keine Anwenderschnittstelle (die mit viel Aufwand selbst programmiert werden muß).

Die britische Firma Prospero Software bietet eine Bibliothek von Grafikroutinen, die mit GSX arbeiten. Sie verzichten auf Skalen, Koordinatenmaps und Überläufe, die dem GSX-Programmierer das Leben schwer machen. Prospero liefert Compiler für ISO PASCAL und FORTRAN 77. Die GSX-Module werden in Programme dieser Sprachen eingebunden und laufen auf jeder Maschine mit installiertem GSX. Sie enthalten Treiber für Maus, Grafiktabletts und Tastatureingaben, aber auch für Drucker, Plotter, VDUs etc. Die GSX-Module plotten Punkte, Linien, Schattierungen und – sofern die Hardware vorhanden ist – auch mit unterschiedlichen Textskalen, Linienbreiten, Farben und Rotation.

GKS und andere Grafikschnittstellen werden übertragbar, indem sie die Anwenderkoordinaten in Gerätekoordinaten umsetzen, die die Hardware „verstehen" kann. In diesem Konzept sind nur die Gerätetreiber maschinenabhängig.



Doch selbst mit einer maschinenunabhängigen Grafikschnittstelle und der relativ bedienerfreundlichen Programmbibliothek von Prospero ist die Erstellung von Grafik immer noch sehr aufwendig. Für den User ist jedoch nur die Anwendung wichtig, ganz gleich, ob Grafik eingesetzt wird, die von GKS, GSX, Prospect, FORTRAN, PASCAL oder anderen Systemen gesteuert wird.

In einem kompletten WIMP-System bietet der Rahmen direkten Zugang zu allen Fähigkeiten des Computers, nicht nur zum Grafiksystem. Der Zugang sollte weiterhin nicht nur über Spezialmodule oder Betriebssystemroutinen möglich sein. Mit GEM und MS-Windows haben Digital Research und Microsoft diesen Weg eingeschlagen. Außer IBM arbeiten auch Apricot und RML (Nimbus) mit GEM und MS-Windows.

Das grafische Kernsystem, auf dem GEM aufgebaut, enstand aus dem Bedarf für übertragbare Grafiksoftware. Mit GSX hatte Digital Research eine kleinere Version von GKS geschaffen, die zwar dem Standard nicht entsprach, dafür aber billig war. Inzwischen wurde das Konzept von DR zum „Graphics Environment Manager" (GEM) weiterentwickelt. GEM ist daher das Ergebnis zweier Entwicklungslinien – zum einen des GKS-Standards und zum anderen des auf SMALLTALK aufgebauten „Schreibtischsystems", das durch den Macintosh von Apple bekannt wurde.

Arbeitsplatz mit Mäusen

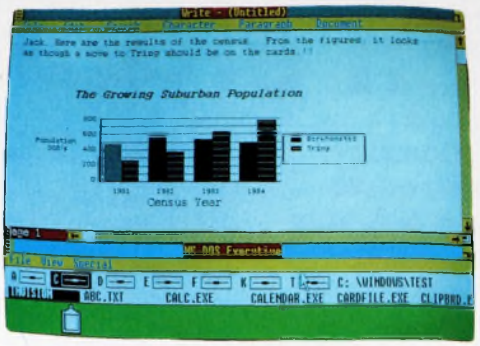
Ebenso wie die Anwendersoftware mit dem Betriebssystem CP/M unterschiedliche Hardware steuert, bietet GEM den Rahmen für die Maschinensteuerung eines WIMP-Systems. Eins der Grundkonzepte ist dabei der „Arbeitsplatz", der den Bereich von der Maus bis zum vollständigen Grafikterminal umfaßt.

GEM spricht jede Systemkomponente auf die gleiche Weise an, die Hardwareeigenschaften und Einschränkungen wurden in den Gerätetreibern der einzelnen Arbeitsplätze versteckt. So kann beispielsweise jedes Ausgabegerät – ob Plotter oder VDU – angewiesen werden, eine bestimmte Form zu zeichnen. Da die Hardwareeigenschaften jedoch Grafikfähigkeit und Auflösung stark beeinflussen, müssen die Grafikanwendungen bei geräteunabhängiger Steuerung radikal von der Hardware isoliert werden..

Jedes Gerät hat seinen eigenen Satz „Gerätekoordinaten". Farbmonitore arbeiten beispielsweise je nach Grad der Bildschirmauflösung auf den X- und Y-Achsen mit unterschiedlichen Maßstäben. Die Geräteeigenschaften und Koordinaten sind jedoch die einzigen Parameter, die sich in einem System, das mit GEM arbeitet, verändern.

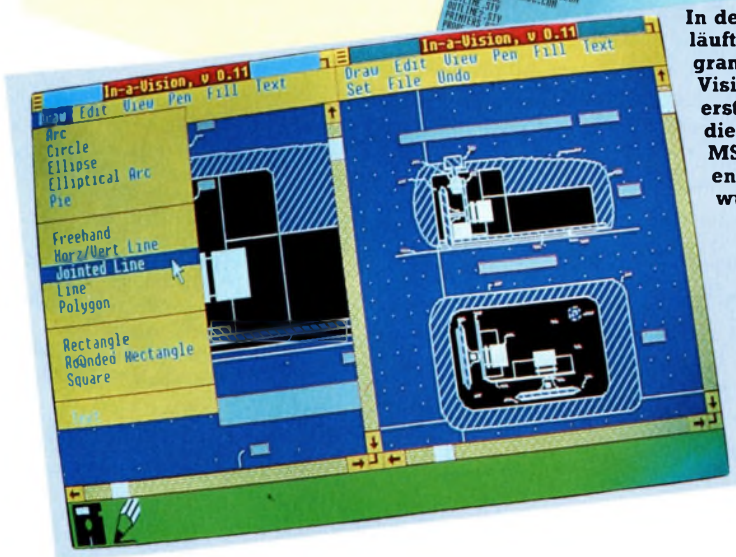
Gerätekoordinaten werden von einem Satz „normaler" Koordinaten abgeleitet, die GKS für alle Grafikvorgänge einsetzt. Diese genormten Gerätekoordinaten (NDC – Normalised Device Coordinates) werden wiederum in die Koordinaten der Anwendungssoftware – die sogenannten „World Coordinates" (WCs) – übernommen, die der Anwendungsprogrammierer auf jede beliebige Skala festlegen kann. Die NCD-Ebene dieses dreifachen Systems wird auf eine Skala von reellen Zahlen im Bereich von Null bis Eins übertragen. Der Auflösungsgrad findet nun nur noch in der Arithmetik der reellen Zahlen eine Einschränkung.

Die dabei entstehenden Schwierigkeiten werden von Computern der Groß-EDV durch hohe Geschwindigkeit und große Kapazitäten ausgeglichen. Auf Microcomputern ist das jedoch nicht so einfach. Da die Arithmetik der reellen Zahlen im Vergleich zur Ganzzahlenverarbeitung viel Zeit braucht, weicht GEM in dieser Hinsicht von GSK ab. Der NDC verfügt in GEM über eine Skala von Null bis 32767, so daß jeder Wert in einem 16-Bit Maschinenwort untergebracht und verarbeitet werden kann und Rechenzeit für Umwandlungen auf einem Minimum bleibt.



Farben können bei der Bearbeitung von Bildern und der Betonung von Meldungen viel ausmachen. Bei der Übernahme eines von einem Grafikprogramm entwickelten Diagramms in eine Textverarbeitung werden sie als Grauwerte dargestellt.

Eine Grafikumgebung unterstützt auch alle normalen Systemzugriffe. Das Bild zeigt das DIRectory des augenblicklich aktiven Laufwerks.



In den Fenstern läuft das Programm „In-a-Vision", die erste Software, die speziell für MS-Windows entwickelt wurde.

MS-Window ist eine von Microsoft entwickelte Grafikumgebung, die speziell auf die Kompatibilität mit anderen Microsoftanwendungen ausgerichtet wurde. Die Bildschirminhalte zeigen den Einsatz des Systems. Im Gegensatz zum Macintosh von Apple arbeiten GEM und MS-Window mit Farbe.

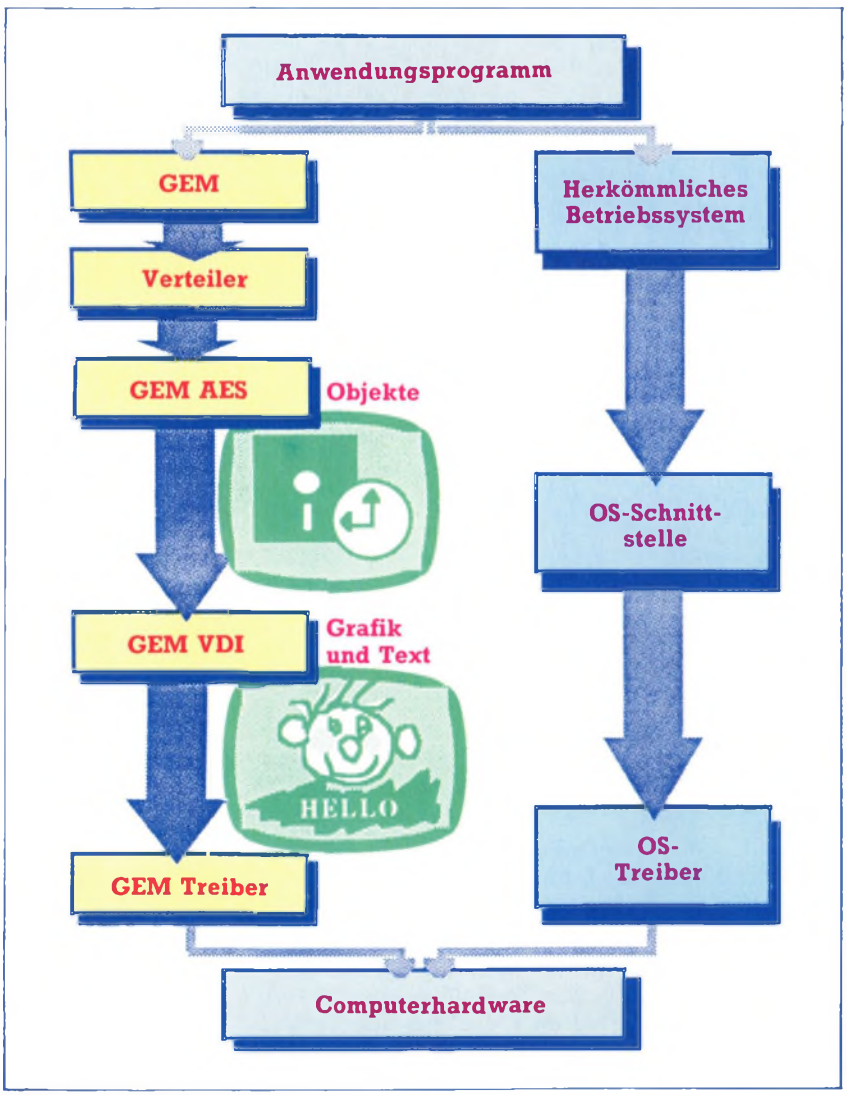
Arbeitsplatz am offenen Fenster

In der vorigen Folge erfuhren wir, daß GEM durch Verwendung des internationalen GKS-Standardrahmens einen hohen Grad an Übertragbarkeit erreicht. Wir sehen uns nun die GEM-Umgebung genauer an und untersuchen Abläufe, Programmstrukturen und Möglichkeiten für den kommerziellen Einsatz.

Die leichte Bedienung von GEM läßt kaum etwas von den außerordentlich komplizierten Vorgänge in seinem Inneren ahnen. Das VDI (Virtual Device Interface) zur Entwicklung von maschinenunabhängigem Code besteht aus zwei Teilen. Ein Teil – das GDOS – spielt die Rolle des herkömmlichen Betriebssystems, und bearbeitet im wesentlichen Eingaben des Anwenders. Der zweite Teil (die Gerätetreiber) ist auf die aktuelle Hardware eingestellt.

Während „Microsoft Window" fast ausschließlich bei kommerziellen Computersystemen eingesetzt wird, findet man GEM auch bei kleineren Systemen. Bei dem neuen Atari 520ST (mit dem Motorola 68000) gehört GEM bereits zu Lieferumfang, und auch auf dem Apricot und dem RML Nimbus ist es verfügbar.

Erfahrene Anwendungsprogrammierer programmieren normalerweise nicht für eine bestimmte Maschine, sondern für ein Betriebssystem. WIMPs, die auf GKS aufbauen (darunter auch GEM), sprechen ihre Grafikkomponenten nach einem festen Standard an. Unter GEM bzw. GKS entwickelte Programme sind damit theoretisch auf jede Hardware übertragbar, die die entsprechenden Kernroutinen für Grafik besitzt. GEM, GSX und GKS bieten einen Softwarerahmen, der sich um das bestehende System und die Grafikhardware legt. Dabei steuert das maschineneigene Betriebssystem ganz normal Tastatureingaben und Dateiverwaltung, während der zusätzliche Softwarerahmen die Grafik übernimmt.



Obwohl Digital Research sein Produkt auch GEM DOS nennt, führt GEM selbst keine der üblichen DOS-Funktionen aus. Befehle, die nicht per Tastatur kommen, werden in normale Betriebssystemaufrufe umgewandelt, die Aktualisierung der Bildschirmanzeige oder anderer Grafikperipherie dagegen steuert das darunterliegende GSX-System – das Herz von GEM. Dieser Vorgang läuft zusätzlich zu dem herkömmlichen Befehlszeileninterpreter ab.

Bei jedem Ablauf, der mit Anzeigedaten arbeitet, wird der normale zeichenweise Bildaufbau durch einen Grafikstrom mit Bit-Map Format ersetzt, der dem entsprechenden Gerätetreiber Daten liefert. Jede Hardwarekomponente besitzt einen eigenen Treiber, der ihre Geräteeigenschaften (Eingabe, Ausgabe, Auflösung, Farbe etc.) berücksichtigt.

Ein Fenster für „Hallo"

Die Programmierung unter GEM ist recht kompliziert. Ohne Standardmodule, die Ereignisse steuern, Fenster aktivieren oder deaktivieren (und darunterliegende Inhalte wiederherstellen) etc., muß der GEM-Programmierer jede Routine aus GEM/GSX VDI Grundelementen aufbauen. So ist unter GEM beispielsweise der Aufbau eines Fensters mit einem einfachen Text, wie PRINT „Hallo!" in BASIC schon eine umfangreiche Programmieraufgabe. Zunächst wird die Größe und Position des Fensters mit Standardwerten initialisiert. Diese Werte müssen veränderbar sein, damit das Fenster ver-

schoben oder auf eine andere Größe gebracht werden kann. Weiterhin muß der Programmierer Attribute wie Text- und Hintergrundfarben, Schriftart, Schriftgröße etc. bestimmen.

Die Aktivierung eines anderen Fensters ist ein „Ereignis", das den aktuellen Ablauf überlagern kann. Die Ausführung wird dann entweder für die Dauer der Überlagerung gestoppt oder läuft im Hintergrund weiter (MS-Windows, Concurrent DOS und die neue Multitasking-Version von GEM). Bei der Reaktivierung sollte der Vorgang wieder als oberes Fenster auf der Anzeige stehen und mit seinem Inhalt die anderen Bildschirmfenster überlagern.

Jedes Programm – auch das kleinste – muß den gesamten Code für die eigene Fenstersteuerung enthalten, darunter Routinen, die ständig den aktuellen Inhalt des Fensters und seine Position in Bezug zu nicht dargestellten Daten feststellen. Das bedeutet aber auch, daß Verwaltungsroutinen rund zehn KByte des Quellentextes belegen, bevor das Programm überhaupt etwas ausführen kann.

Für GEM oder GSX gibt es keine Standardprozeduren, die diese Verwaltungsroutinen erledigen. Softwarehäuser, die mit GEM arbeiten, programmieren in MODULA-2, PASCAL, C, BCPL oder Assembler, haben aber trotz all ihrer Erfahrung zuweilen immer noch große Probleme mit GEM. Wenn Sie vorhaben, in ein paar hundert BASIC-Zeilen eine einfache GEM-Anwendung für Ihren Heimcomputer zu schreiben, können Sie davon ausgehen, daß diese Aufgabe etwas Zeit erfordern wird. Wenn Sie jedoch in einer der erwähnten Sprachen programmieren können, steht Ihnen eine Reihe von Produkten zur Verfügung, die brauchbare Programmierschnittstellen zu dem darunterliegenden WIMP-System bieten.

Mehrere GEM-Angebote

Die englische Firma TDI brachte einen der ersten einsatzfähigen „Tool Kits" für WIMP auf den Markt. Der MODULA-2/ST bietet den Besitzern eines Atari 520ST die Möglichkeit, unter GEM in der Sprache zu programmieren, die von Nikolaus Wirth für Lilith entwickelt wurde.

Für ernsthafte GEM-Anwender stehen mehrere Systeme zur Verfügung:

- Ein Atari 520ST (mit GEM), MODULA-2 und dem TDI Toolkit sind schon recht günstig zu haben;
- Ein Apricot, GEM und ein guter MODULA-2, PASCAL, BCPL oder C-Compiler kosten etwas mehr;
- Ein CP/M–86 oder MS-DOS System mit GSX, Pro-PASCAL und der Prospect Modulbibliothek für die Grafikschnittstelle sind zwar teuer, aber erprobt und vielseitig einsetzbar;
- Der IBM PC, AT oder ein kompatibles Gerät, ein 8086 Assembler oder C und die GEM-Programmierhilfen von Digital Research eignen sich für erfahrene Programmierer mit Systemkenntnissen und viel Zeit und Geld.

Allen, die keine Zeit zum Programmieren haben, bietet Digital Research eine Reihe von Anwendungen, die mit der GEM-Umgebung arbeiten. GEM Desktop ist eins der „Schreibtischsysteme", die (mit der neuen Multitaskingversion von GEM) mehrere Anwendungen steuern, Dateien verwalten, drucken, plotten und die Funktionen der Echtzeituhr und des „Taschenrechners" übernehmen. GEM Write ist eine WIMP-ähnliche Textverarbeitung, GEM Paint ein Zeichenpaket für den künstlerischen Bereich. Jedes der Systeme ist einzeln oder im Paket erhältlich.

Die Gesichter von GEM

GEM besteht aus zwei Hauptteilen: dem AES „Applications Environment Services" (Steuerung der Anwendungsumgebung) und dem VDI „Virtual Device Interface" (virtuelle Geräteschnittstelle). GEM und AES enthalten eine Bibliothek von Subroutinen (mit Rahmenmodulen für Fenstersteuerung, Eingaben per Maus, Meldungsanzeige und Objektgrafik) und einen Verteiler mit Multitasking – im Augenblick lassen sich drei Vorgänge gleichzeitig steuern, die neue GEM-Version ist für zwölf Vorgänge ausgelegt.

Das VDI des GEM enthält im wesentlichen das aus GKS hervorgegangene GSX ‚Grafics System Extension' (Grafik Systemerweiterung), das mit RASTER-OPS und FACES erweitert wurde. RASTER-OPS sind bitweise logische Vorgänge (AND, OR, XOR) mit Quellen- und Datenblöcken, während FACES Schriftarten in dynamisch ladbare Dateien speichert.

Diese maschinennahen Module sind zweiteilig. Das GDOS „Graphics Davice Operating System" (Betriebssystem für die Grafikperipherie) entspricht dabei der normalen Betriebssystemschnittstelle. Den zweiten Teil bilden die austauschbaren Gerätetreiber.

Das GDOS enthält alle grundlegenden hardwareunabhängigen Grafikfunktionen (die separaten Treiber steuern die einzelnen Gerätekomponenten). Ebenso wie herkömmliche Betriebssysteme in ihren Anwendungsprogrammen den systematischen Speicherzugriff etc. steuern, so bietet auch GDOS ein standardisiertes und übertragbares Grafiksystem, das unabhängig von der eingesetzten Hardware arbeitet.

Eine einzige Subroutine spricht das VDI mit folgenden fünf Argumenten an:

- **Steuerarray**
- **Array der Eingabeparameter**
- **Eingabearray für Punktkoordinaten**
- **Array der Ausgabeparameter**
- **Ausgabearray für Punktkoordinaten**

In der Programmiersprache C gibt es Verbindungselemente, die als mnemotische Kürzel dienen und die Komplexität der zahllosen GEM-Module vereinfachen. So sprechen beispielsweise alle Funktionsaufrufe, die mit v_ anfangen, VDI Routinen an.

Die Klassiker

Klassische Sprachen wie FORTRAN und COBOL haben noch immer viel Einfluß auf die modernen Programmtheorien. Wir beschäftigen uns mit ihrer Entstehung und den ersten wichtigen Konzepten.

Die meisten Menschen empfinden die Programmierung eines Computers als einen ganz normalen Vorgang. Selbst wenn sie damit nur wenig Erfahrung haben, wissen sie zumindest, worum es geht. Um 1950, als das Konzept der Programmierung noch kaum entwickelt war, sah es jedoch völlig anders aus. Dennoch stammt aus dieser frühen Periode eine Reihe von Programmiersprachen, die noch heute im Einsatz sind.

Wir untersuchen, wie sich diese Sprachen zu ihrer jetzigen Form entwickelten, verfolgen ihren Einfluß auf modernere Sprachsysteme und sehen uns die Wirkung auf die Microcomputer der heutigen Zeit an. Doch zunächst: Wie konnte in den frühen Tagen des Computers die Idee einer „Computersprache" überhaupt entstehen?

Programme aus Zahlen

Wenn wir die ersten Entwicklungsstadien beiseitelassen, in denen eine „Programmierung" durch Umlöten der Kontakte geschah, dann bestanden die ersten echten Programme nur aus Zahlen. Geschrieben wurden die Zahlenfolgen üblicherweise in Oktal (Basis Acht). Die Nummern mußten mühsam über Schalter an der Vorderseite des Gerätes und später über Lochkarten oder Lochstreifen eingegeben werden. Jedes neue Programm wurde auf niedrigster Ebene von Grund auf neu entwickelt, und oft mußten die Programmierer die gleichen Routinen wieder und wieder schreiben. Nach und nach entstand das Konzept einer Bibliothek für Subroutinen – den Begriff „Subroutine" gab es damals allerdings noch nicht. Diese „Bibliotheken" fanden sich damals in den Notizbüchern der Programmierer und wurden je nach Bedarf in neue Programme abgeschrieben. Die Notizbücher waren sorgsam gehütete Geheimnisse, zu denen andere Programmierer nur selten Zugang erhielten.

In Manchester geschah ein großer Schritt nach vorn, als 1951 das EDSAC-System gebaut wurde und Wheeler, Wilkes und Gill einen aufeinander abgestimmten Satz von Subroutinen entwickelten, der zum Lieferumfang der Maschine gehörte. Da nun alle mit den gleichen Subroutinen arbeiteten, war auch die Programmentwicklung einfacher, und es zeigten sich die ersten strukturellen Ähnlichkeiten mit modernen Abläufen. Die Codeblocks führten bestimmte Aufgaben durch den Aufruf von Subroutinen aus, die Standardfunktionen wie Ein-/Ausgabe und Berechnungen steuerten.

Mit wachsenden Speicherkapazitäten ließen sich die Subroutinen auch intern – oder zumindest im direkten Zugriff – speichern. Von hier war es nur noch ein kleiner Schritt zur Entwicklung eines Codes, in dem der Programmierer die gewünschten Subroutinen mit alphabetischen Zeichen oder mathematischen Formeln angeben konnte. Das Programm „sah" sich diesen Code Zeile für Zeile an, rief die erforderlichen Subroutinen auf und bearbeitete dann die nächste Zeile. Mit dieser Technik arbeiten auch heute noch die modernen BASIC-Interpreter. Eines der ersten Systeme dieser Art war der „Short Code", den John Maunchly 1949 auf dem BINAC-Computer und später dem UNIVAC entwickelte.

In einer weiteren Verbesserung dieser frühen Technik wurden die Subroutinen vor ihrer Ausführung erst zu einem Ausgabegerät geschickt und so das Programm gespeichert. In diesem Stadium entstanden dann auch die ersten Compiler („compilieren" – zusammentra-

Meilensteine

Eine einfache Chronologie der frühen Programmiersprachen:

1951 Das erste Programm wird digital in dem EDSAC-Computer (Manchester) gespeichert

1952 Der „Short Code" von John Maunchly

1953 „Speedcoding" von Laning und Zierly

1954 Der A-2 Compiler
Die ersten FORTRAN-Beschreibungen

1955 Die ersten echten Compiler für FORTRAN und FLOW-MATIC

1956 Allgemeine Veröffentlichung der ersten FORTRAN- und FLOW-MATIC-Compiler
Beschreibung für FORTRAN II

1957 Definition und Veröffentlichung des ersten ALGOL-Compilers

1958 Veröffentlichung von FORTRAN II

1959 Erste Beschreibung von COBOL
Erster LISP-Compiler

1960 Veröffentlichung des ersten COBOL-Compilers und des überarbeiteten ALGOL 60

Einfache Reservierungssysteme

FORTRAN IV:

```
C       THEATER-PLATZ-RESERVIERUNG
C       DAS PROGRAMM NIMMT EINE SITZ-
C       NUMMER ENTGEGEN, PRUEFT, OB ER
C       BEREITS GEBUCHT IST
C       BUCHT IHN, WENN ER FREI IST, ODER
C       GIBT EINE MELDUNG AUS.
C
C       VARIABLEN DEKLARIEREN
C
        INTEGER SEATNO, SEAT (500)
C
C       ALLE FREIEN SITZE MARKIEREN
C
        DO 100 I = 1,500
100     SEAT (I) = 0
C
C       SITZNUMMER LESEN UND VERFUEG-
C       BARKEIT PRUEFEN
C
101     READ (1,10) SEATNO
        IF (SEAT (SEATNO). NE.0) GOTO 102
C
C       SITZ IST FREI
C
        SEAT (SEATNO) = 1
        WRITE (1,20)
        GOTO 103
C
C       SITZ IST NICHT FREI
C
102     WRITE (1,30)
C
C       NAECHSTE SITZNUMMER
C
103     GOTO 100
C
C       AUSGABE FORMATIEREN
C
10      FORMAT (I4)
20      FORMAT (1H, 13HSITZ IST FREI)
30      FORMAT (1H, 24HSITZ IST BEREITS GE-
        BUCHT)
        END
```

ALGOL 60:

```
comment, THEATER-PLATZ-RESERVIERUNG
DAS PROGRAMM NIMMT EINE SITZNUMMER
ENTGEGEN, PRÜFT, OB ER BEREITS GEBUCHT
IST
BUCHT IHN, WENN ER FREI IST, ODER GIBT EINE
MELDUNG AUS.
BEACHTEN SIE, DASS ALGOL KEINE EIN- ODER
AUSGABEBEFEHLE ENTHÄLT:
begin:
   integer SEATNO, I;
   integer array SEAT[1:500];
   comment ALLE FREIEN SITZE MARKIEREN
   for I:= 1 step 1 until 500 do
      SEAT [1]:=0
   comment SITZNUMMER LESEN UND VER-
   FÜGBARKEIT PRÜFEN;
NEWSEAT: ((read SEATNO));
   if SEAT [SEATNO] = 0 then begin
      SEAT [SEATNO]: = 1;
      ((print 'SITZ IST FREI'));
   end
   else
      ((print 'SITZ IST BEREITS GEBUCHT'));
      goto NEWSEAT;
   end
```

COBOL 74
(Nur Daten- und Verarbeitungsteil):

```
* THEATER-PLATZ-RESERVATIONEN.
* DAS PROGRAMM NIMMT EINE SITZNUMMER
ENTGEGEN, PRÜFT, OB ER BEREITS GEBUCHT
IST,
*BUCHT IHN, WENN ER FREI IST, ODER GIBT
EINE MELDUNG AUS.

DATA DIVISION.
WORKING STORAGE SECTION.
01 SEAT-AVAILABILITY.
   02 SEAT PIC 9 OCCURS 500 TIMES.
77 SEAT-NUMBER PIC 999.
77 LOOP-COUNTER PIC 999.
77 SEAT-AVAILABLE PIC X (13) VALUE
   „SITZ
IST FREI".
77 SEAT-BOOKED PIC X (24) VALUE
   „SITZ IST BEREITS GEBUCHT".
PROCEDURE DIVISION.
MAIN PARAGRAPH.
* ALLE FREIEN SITZE MARKIEREN
   PERFORM MARK-SEAT-AVAILABLE-
      PARAGRAPH
      VARIYING LOOP-COUNTER FROM 1 BY 1
      UNTIL I › 500.
* SITZNUMMER LESEN UND VERFÜGBARKEIT
PRÜFEN
   PERFOM GET-SEAT-NUMBER PARAGRAPH.
   STOP RUN.
MARK-SEAT-AVAILABLE-PARAGRAPH.
    MOVE ZERO TO SEAT (LOOP-COUNTER).
GET-SEAT-NUMBER-PARAGRAPH.
   ACCEPT SEAT-NUMBER.
   IF SEAT (SEAT-NUMBER) IS EQUAL TO 0
      MOVE 1 TO SEAT (SEAT-NUMBER)
   DISPLAY SEAT-AVAILABLE
   ELSE
   DISPLAY SEAT-BOOKED.
   GOTO GET-SEAT-NUMBER-PARAGRAPH.
```

Die folgenden drei Listings in FORTRAN, COBOL und ALGOL zeigen die Programmstruktur und einige vergleichbare Eigenschaften. Jedes Programm nimmt eine Sitznummer entgegen, prüft, ob die Nummer zuvor schon eingegeben (gebucht) wurde. Falls nicht, wird der Sitz als gebucht markiert.

gen – bedeutete ursprünglich, daß Programme ähnlich wie Artikelsammlungen aus vielen Einzelteilen zusammengetragen wurden).

Einer der frühesten erfolgreichen Compiler war der A-2, den 1955 ein von Grace Hopper geleitetes Team bei Remington Rand entwickelte. Der A-2 arbeitete mit „Dreieradressen". Dabei erhielt jeder Vorgang einen mnemotischen Namen, gefolgt von drei Adressen – zwei für die Quellenangabe und eine für die Bestimmung. In einigen Bereichen ähnelte er einem Assembler, war jedoch auf keine bestimmte Maschine angepaßt. Später entstand daraus ARITH-MATIC, gefolgt von AT3 oder MATH-MATIC – eine ähnliche, vom Remington Rand entwickelte Sprache.

Ende 1951 wurde klar, daß auf den Computern Programme liefen, die vom Blickpunkt des Anwenders aus nicht in der Maschinensprache des Gerätes geschrieben waren. Die Tatsache, daß der Computer das Programm erst in seinen Maschinencode übersetzen mußte, war dabei nicht von Bedeutung.

In diesem Fall konnte man aber auch eine völlig neue Sprache entwickeln, die dem Programmierer die Programmentwicklung erleichterte, statt die Übersetzung noch weiter zu beschleunigen. Die neue Hardware wurde sowieso immer schneller und leistungsfähiger, so daß die Programmierer kaum mithielten.

Das nächste Problem trat auf, als sich zeigte, daß kein wie auch immer gearteter Standard existierte. Jede Maschine arbeitete mit ihrer eigenen Sprache, die auf ihren Aufgabenbereich ausgerichtet war. Ohne eine hardwareunabhängige Sprache, die sich für einen weiteren Aufgabenkreis eignete, konnte es nicht weitergehen. Die Anfänge von 1954 führten schließlich zu FORTRAN (das FORmula TRANslation System von IBM), der ersten echten Programmiersprache.

Sprache war gefordert

FORTRAN eignete sich ausgezeichnet für die Zahlenverarbeitung, aus denen die Computerarbeit damals hauptsächlich bestand. Als die Geschäftswelt sich jedoch dafür interessierte, ihre großen Datenmengen von Computern bearbeiten zu lassen, mußte eine Sprache her, die dem Geschäftsenglisch entsprach.

Interessanterweise gab es zu dieser Zeit eine ganze Reihe von Programmierern, die sich nicht vorstellen konnten, daß Computer je Wörter statt Zahlen „verstehen" könnten. Diese Meinung erwies sich jedoch schon bald als falsch, als Grace Hopper (und andere) eine Sprache namens FLOW-MATIC entwickelten, die von Management und Programmierern gleichermaßen verstanden wurde. 1956 wurde daraus COBOL (COmmon Business Oriented Language). Das Sprachsystem war exakt definiert.

Zu diesem Zeitpunkt war die Zahl der Computer und Programmierer bereits stark gewachsen. Einige Forscher beschäftigten sich mit dem theoretischen Aspekt der Programmiersprachen und suchten bessere und elegantere Wege für die Umsetzung von Algorithmen. Dies führte 1958 schließlich zu der Entwicklung von ALGOL (ALGOrithmic Language). ALGOL fand nie die Verbreitung von FORTRAN oder COBOL, nimmt in der Sprachentwicklung aber einen bedeutenden Platz ein. ALGOL förderte als erste den systematischen Programmaufbau, der zu einem der wichtigsten Bestandteile aller neueren Sprachen wurde.

In den frühen Tagen entstand eine Reihe weiterer Sprachen, die teilweise heute noch in Gebrauch sind. Eins der besten Beispiele ist die zwischen 1956 und 1958 entwickelte LISP (LISt Processing Language). Die Sprache gewinnt inzwischen im Feld der Künstlichen Intelligenz eine ständig wachsende Bedeutung. Die Hauptsprachen der letzten zwanzig Jahren bleiben jedoch FORTRAN, COBOL und ALGOL. Sie wurden im Laufe der Zeit mehrmals überarbeitet und erhielten neue Fähigkeiten, die den modernen Trends der Programmentwicklung Rechnung tragen. Die neuesten Versionen sind FORTRAN 77 (die Sprachdefinition von 1977), COBOL 74 und ALGOL 68.

1964 erschien am Dartmouth College (USA) eine vereinfachte (und von ALGOL beeinflußte) FORTRAN-Version, die das Lernen der Sprache erleichtern sollte. Die Version funktionierte außerdem auf den neuen Mehrplatzanlagen, die im Timesharing-Verfahren arbeiteten. Diese Version wurde unter dem Namen BASIC (Beginners All-purpose Symbolic Instruction Code) allgemein bekannt.

Als 1968 der neue ALGOL-Standard vorbereitet wurde, war Niklaus Wirth mit der wachsenden Komplexität der Sprache nicht einverstanden und versuchte, ein einfacheres und eleganteres Konzept durchzusetzen. Inzwischen wird das sehr komplexe ALGOL 68 nur noch in akademischen Bereichen eingesetzt, während die weit einfachere Version von Wirth – PASCAL – weite Verbreitung fand.

Short Code

In dem Short Code von John Maunchly sieht die vertraute BASIC-Anweisung

10 A = B + C

so aus:

10 S0 03 S1 07 S2

Dabei ist 10 die Zeilennummer, und S0, S1 und S2 stellen die Symbole für A, B und C dar. 03 und 07 sind die Verarbeitungscodes für Zuordnung und Addition.
In A-2 lautet der Befehl:

ADD B C A

Funktionsfähig

Wir zeigen, wie MS-DOS-Funktionen von der Assemblersprache des 8088 und von Hochsprachen wie PASCAL oder FORTRAN aus angesprochen werden.

Die Funktionsaufrufe von MS-DOS lassen sich in sechs Gruppen unterteilen:

- Die Ein- und Ausgabe von Zeichen an angeschlossene Standardgeräte
- Dateizugriff (darunter auch die Verwaltung des Inhaltsverzeichnisses)
- Speicherverwaltung
- Ablaufsteuerung
- Vermischte Systemfunktionen
- Spezielle Aufrufe für Microsoft-Netzwerke

Diese Systemroutinen können von jeder Anwendung aus aufgerufen werden und bieten eine einheitliche und umfassende Schnittstelle, die die unterschiedlichen Betriebssystemversionen untereinander kompatibel macht. Aufrufe, die mit absoluten Adressen arbeiten oder Hardwarekomponenten direkt ansprechen, lassen sich nicht immer auf andere MS-DOS-Systeme übertragen und laufen auch nur höchst selten mit verschiedenen MS-DOS-Versionen.

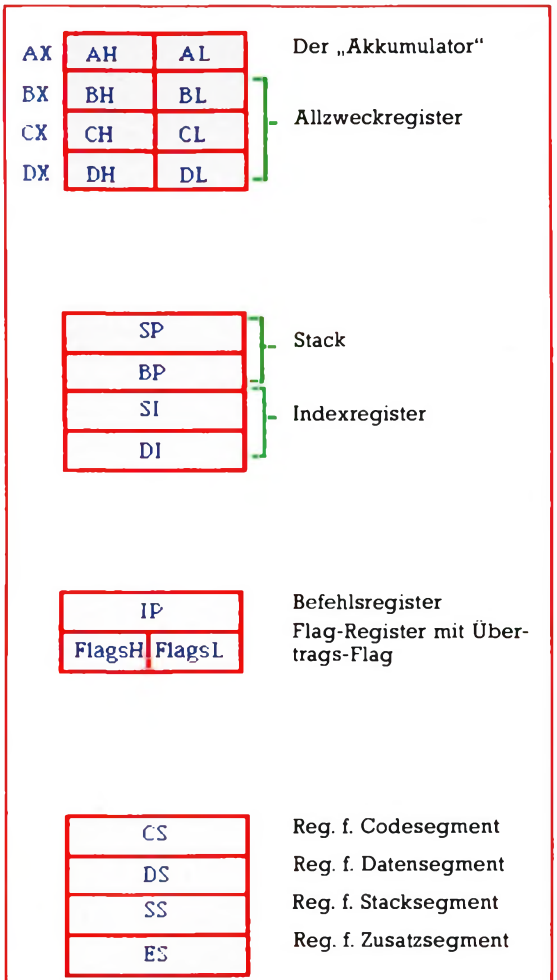
Der Begriff „Funktion", mit der in der MS-DOS-Terminologie Systemaufrufe bezeichnet werden, läßt sich über Unix auf BCPL und C zurückverfolgen. Damit sind jedoch keine Routinen gemeint, die Daten bearbeitet zurückgeben, sondern Vorgänge, die etwa Dateien eröffnen. Derartige Vorgänge sind zwar Prozeduren, werden bei MS-DOS aber mißverständlich Funktionen genannt.

Die größte Bandbreite haben die vermischten Systemfunktionen. Sie arbeiten mit Softwareinterrupts und durchlaufen der Reihenfolge nach folgende Stadien:

1) Alle benötigten Daten in die entsprechenden Register laden (die Laufwerksnummer wird beispielsweise in DL gebraucht).
2) Die Funktionsnummer in AH (das höherwertige Byte von AX) laden.
3) Den Softwareinterrupt 21H auslösen.

Alle von der Funktion zurückgelieferten Daten befinden sich nach der Rückkehr in den Registern des 8086. Die DOS-Funktion 19H (die Funktionsnummern haben Hexadezimalformat) stellt beispielsweise den Code des aktiven Laufwerks in das Register AL. Die Funktion benötigt keinen zusätzlichen Code und läßt sich daher leicht aufrufen:

```
mov ah,19H ;aktuelles Laufwerk feststellen
int 21H    ;Funktion aufrufen
```



Register im Aufruf

Der 8088/8086-Prozessor kann bis zu einem Megabyte adressieren. Er benötigt dafür 20-Bit-Adressen und – da der 8088 nur mit 16-Bit-Adressen arbeitet – zusätzliche Bits zum Aufbau der Adressen. Die Konstrukteure des 8088/8086 umgingen dieses Problem, indem sie das Segment „Offset Format" wählten. Dabei adressiert das Segmentregister einen 64 KByte großen Speicherblock, während das Offsetregister das Byte angibt.

Die wichtigsten Register sind A, B, C, D und das Flag-Register. Sie werden entweder mit dem Zusatz ‚X' als vollständige 16-Bit-Worte angesprochen (z. B. AX) oder als nieder- und höherwertige Bytes mit den Zusätzen ‚L' und ‚H'.

Das Ergebnis steht nun im Register AL. Es stellt das Laufwerk durch einen Zahlenwert dar (A=1, B=2 etc.). Wenn Sie daran den ASCII-Code des Zeichens „A" anhängen und die DOS-Funktion 05H aufrufen (den Code von DL als ASCII-Zeichen darstellen), wird das aktuelle Laufwerk als Großbuchstabe dargestellt:

```
add al,'A'  ;0 in 'A', 1 in 'B' etc. wandeln
mov dl,al   ;in das Register DL stellen
mov ak,05H  ;Zeichen darstellen
int 21H     ;Funktion aufrufen
```

Die im vorigen Beispiel eingesetzten mnemotischen Kürzel entsprechen dem Microsoft-Assembler, der von den OEM-Firmen aber nicht immer mitgeliefert wird. Glücklicherweise ist ein Assembler nicht unbedingt nötig, da MS-DOS sich auch von einer Hochsprache aus leicht aufrufen läßt. Wenn Sie viel in Assembler programmieren, sollten Sie beim Kauf sicherstellen, daß Ihr Assembler verschiebbare Maschinencodedateien im Standard Intel Format (.OBJ) erzeugt. Dieses Format gilt als Industriestandard für Maschinen der 8086-Familie. Mit den meisten Linkern der bekannteren Hersteller (für PASCAL, FORTRAN, C etc.) können Sie die in diesen Sprachen erstellten Module (unter Verwendung bereits assemblierter Standardroutinen im Maschinencode) zu Programmen zusammenstellen.

Assemblieren von Hand

Statt eines Assemblers können Sie auch „handassemblierten" Code verwenden, den Sie mit dem BASIC-Befehl POKE in einen freien Speicherblock setzen. Dieser Ablauf eignet sich jedoch nur für einfache Aufgaben (wie die eben beschriebenen). Sie brauchen dazu eine Liste der Intel Opcodes, eine ausführliche Memory-Map der Maschine, die Information, welche Speicherbereiche zur Verfügung stehen – und viel Geduld. Da der so entstandene Code nur schwer lesbar ist, sollten Sie diese Methode aber möglichst vermeiden. Praktischerweise enthalten die meisten compilierten Sprachsysteme für MS-DOS gleich alle nötigen Erweiterungen für Systemroutinen, so daß nicht erst externer Maschinencode geschrieben werden muß.

Das von Microsoft entwickelte PASCAL-86 kann mit der Prozedur DOSXQQ direkt auf Funktionen zugreifen, und auch MT+86 von Digital Research besitzt eine ähnliche Routine. Vielleicht die beste (mit Sicherheit aber die einfachste) Lösung bietet Prospero Software in dem weit verbreiteten Compiler im ISO Standard (Pro PASCAL) und dem preiswerten Turbo PASCAL. Beide Systeme sind eine ausgezeichnete Grundlage für die DOS-Programmierung ohne Assemblersprache und helfen dem Einsteiger weiter.

Pro PASCAL und Turbo PASCAL besitzen eine TYPE Definition, die alle wichtigen Register des 8086 zu einem RECORD zusammenstellt. Dessen Felder werden vor dem Aufruf initialisiert und enthalten danach die von den DOS-Funktionen gelieferten Daten. Die Syntax ist bei jedem Aufruf gleich (die Handbücher sollten zu Rate gezogen werden, denn sie enthalten eine genaue Beschreibung).

Prospero bietet weiterhin eine Prozedur namens system, die mit einem Variablenparameter dieser Art (SysReg) den Funktionsaufruf durchführt. Die Prozedur zur Anzeige des aktuellen Laufwerks lautet daher in PASCAL:

```
WITH register DO
  BEGIN {aktuelles Laufwerk anzeigen}
    AH:=25; {Funktionscode 19H}
    system ( register ); {aufrufen }
    WriteLn ( 'Das aktuelle Laufwerk ist ',
    chr ( AL + ord ( 'A') ),':')
  END
```

Eine vollständige Funktion

Vor dem Speichern von Daten sollten Sie feststellen, wieviel Platz auf der Diskette noch zur Verfügung steht, da das Speichern auf eine fast volle Diskette unangenehme Folgen haben kann. Über den Aufruf der DOS-Funktion 36H (54 dezimal) prüfen Sie die Diskette vor dem Schreiben und können dann entweder die Datei rechtzeitig schließen oder eine andere Diskette einlegen.

Für den Systemaufruf wird die Funktionsnummer wie gewohnt in AH gestellt und die Laufwerksnummer in DL (Laufwerk A ist 1, B ist 2 etc.). Code 0 gibt das Standardlaufwerk an. Sie müssen nun nur noch das DL-Feld mit dem entsprechenden Wert initialisieren, 36H in AH speichern und die Systemfunktion aufrufen. Nach dem Rücksprung enthält AX entweder den Fehlercode 0FFFFH (beispielsweise) oder folgende Register die gültigen Daten:

AX Die Anzahl der Sektoren pro „Cluster"
BX Die Zahl der verfügbaren „Cluster"
CX Die Anzahl der Bytes pro Sektor
DX Die Gesamtzahl der „Cluster"
Die Bytezahl der freien Diskettenkapazität errechnet sich aus folgenden Werten:
Cluster * Sektoren pro Cluster * Bytes pro Sektor
Cluster sind Speicherbereiche, die eine Reihe vollständiger Sektoren umfassen.

```
FUNCTION DiskSpace ( Laufwerk : integer )
  : integer;
  { liefert die Zahl der freien Diskettenbytes }
  VAR
    register : SysReg; { siehe Handbuch }
  BEGIN
    WITH register DO
      BEGIN
        DL := Laufwerk; { 0 = Standard,
              A = 1 etc. }
        AH := 36H; { Funktionsnummer }
        system ( register ); { aufrufen }

        IF AX = OFFFFH
          THEN { ein Fehler ist aufgetreten }
            BEGIN
            WriteLn ( 'DiskSpace : ERROR
            ( ungültiger Code ? )');
            DiskSpace := 0 { mehr wissen
              wir nicht }
          END
        ELSE
            DiskSpace := AX * BX * CX
    END { Sektoren * Cluster * Bytes }
  END; { DiskSpace }
```

Kompaktklasse

Aus den im vorhergehenden Kapitel behandelten Verkürzungstechniken soll ein vielseitiges Programm zur „Textkompression" entwickelt werden. Unser Programmvorschlag arbeitet mit Vier-Bit-Blöcken. Den Assembler-Teil finden Sie in dieser Ausgabe. Weitere Ergänzungen dazu folgen im nächsten Heft.

Die bei unserem Programm eingesetzte Methode ist eine verkürzte Fassung des Vier-Bit-Algorithmus. Der Text wird dabei in Vier-Bit lange Blöcke umgesetzt, die entweder eines der am häufigsten verwendeten Zeichen oder einen Code mit Informationen über den folgenden Vier-Bit-Block darstellen. In der Tabelle rechts wird erläutert, was die einzelnen „Nibbles" (Vier-Bit-Blöcke) bedeuten.

In der vorliegenden Form komprimiert das Programm nur Texte aus Großbuchstaben, Leerzeichen, Kommas und Punkten. Es können aber auch bis zu 16 Tokens (Abkürzungen) verarbeitet werden.

Beim Einsatz muß der Anwender die Adresse des zu komprimierenden Strings eingeben und auch die Adresse, an der der neue String gespeichert werden soll. Der String wird von einem Zählbyte eingeleitet, auf das die Buchstaben folgen. Die Stringlänge ist auf 255 Bytes begrenzt.

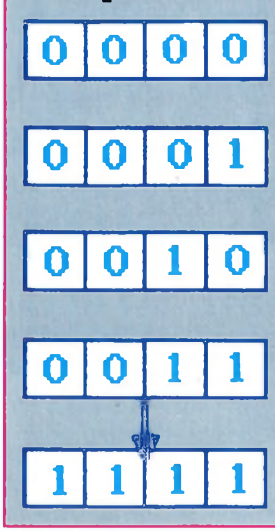
Wer in Maschinensprache programmieren kann, wird mit der Erweiterung keine besonderen Schwierigkeiten haben. Eine wirkungsvolle Ausbauvariante könnte etwa die Erzeugung einer Token-Tabelle mit 255 Elementen sein. Damit könnten neben Groß- und Kleinbuchstaben auch zusätzliche Tokens verarbeitet werden. Wenn dies vor der Prüfung auf Acht-Bit-Tokens geschieht, kann das restliche Programm bis auf die zusätzliche Tabelle unverändert bleiben.

Die Tabelle für diese Variante sollte so aufgebaut sein, daß die Vier-Bit-Werte die häufigsten Buchstaben in Kleinschrift sowie das Leerzeichen und das Symbol für „neue Zeile" darstellen. In der Acht-Bit-Tabelle stehen dann nur die weniger gebräuchlichen Kleinbuchstaben. Seltene Kleinbuchstaben stehen zusammen mit den Großbuchstaben und Tokens in der neuen Tabelle.

Die Programmadresse ORG kann je nach Rechner abgewandelt werden. Das Programm muß assembliert und der übersetzte Code auf Cassette oder Diskette gespeichert werden.

Unser Programm übersetzt Text in Vier-Bit-Blöcke, sog. „Nibbles". Die einzelnen Nibbles haben unterschiedliche Bedeutung (siehe Tabelle). Hier sind nur die Werte 0 bis 2 eingesetzt. Der Einsatz des Wertes 3 könnte beim Erweitern des Programms ein zusätzliches Signal definieren. Die 3 würde dabei als Zeiger einer Tabelle mit 255 Elementen fungieren. Dadurch wird die Darstellung von Kleinbuchstaben und eine größere Zahl von Tokens möglich.

Kompressions-Codierung



Programm zur Textkompression

Unser Programm für die Textkompression erscheint in zwei Abschnitten. In dieser Folge haben wir das Assembler-Listing für Z80-Rechner abgedruckt. Die Version für den Prozessor 6502 folgt im nächsten Heft. Außerdem versorgen wir Sie auch mit BASIC-Ladeprogrammen für beide Versionen. Das Programm läuft, auch wenn Sie keinen Assembler haben.

```
        org 30000

        jr      start           ; leap round the data space

string: dw      0       ; load with input string address
output: dw      0       ; enter address for output string
status: db      0       ; status return, zero signals ok
mask:   db      0       ;
len:    db      0       ; storage space for input length

start:  ld      hl,(string)     ; get address of string to
                                ; compress
        ld      a,(hl)          ; get input string length
        ld      (len),a         ; store it away
        inc     hl              ; point at start of string
        ld      de,(output)     ; point at output space
        push    de              ; save start of output
        inc     de              ; leave 1 space for count
        ld      (output),de     ; save output address till
                                ; we get something to store
        ld      a,255
        ld      (mask),a        ; handle left nibble first

nchar:  ld      a,(hl)          ; get byte to work on
        call    check           ; make sure it's within range
        jr      nz,badchar      ; jr if character not allowed
        call    token           ; check if start of a token
        jr      z,gottoken      ; found a token so process it
        call    fourbit         ; check for a 4 bit character
        jr      z,got4bit       ; character is 4 bits so jump
        call    eightbit        ; get 8 bit character value
        push    af              ; no need to check - save value
        ld      a,8             ; while '8 bit char' indicator..
        call    writenib        ;..output
```

```
        pop     af              ; restore 2nd nibble of 8 bit val
got4bit:call    writenib        ; output 2nd nibble of 8 bit, or
                                ; 1st nibble of 4 bit, value
rejoin: inc     hl              ; point at next input character
        ld      a,(len)         ; get back current length
        dec     a               ; minus one for char just proc'd
        ld      (len),a         ; put back new length
        and     a
        jr      nz,nchar        ; jr to proc next char if exists
        ld      (status),a      ; signal finished ok, a will = 0
        ld      a,2             ; 2 = end of compressed text
        call    writenib        ; write it out
        ex      de,hl           ; final output address into hl
        pop     de              ; get output string address
        ld      a,(mask)        ; get mask value
        and     a
        jr      z,nodec         ; if mask zero then final value ok
        dec     hl              ; ignore final byte, it's null
nodec:  and     a               ; clear carry for subtraction
        sbc     hl,de           ; get length
        ld      a,l             ; get length - not more than 255!
        ld      (de),a          ; store count at begin of output
        ret                     ; return

badchar:pop     de              ; clear stack
        ld      a,255
        ld      (status),a      ; signal failure
        ret

gottoken:
        push    af              ; save token nibble
        ld      a,1             ; signal token coming
        call    writenib
        pop     af              ; restore nibble
        call    writenib        ; output token
        jr      rejoin

        ; subroutine to check for start of new token
token:  push    hl              ; save input address
        ex      de,hl           ; put input address into de
        ld      hl,tktable      ; point hl at table of tokens
        ld      a,(len)         ; get remaining length into a
        ld      c,a             ; and transfer it to c
        ld      b,0ffh          ; initialize token number
tok1:   ld      a,(hl)          ; get length of current token
        and     a
        jr      z,notfound      ; if length=zero then endoftable
        push    de              ; save these pointers in case
        push    hl              ; they are needed
        inc     hl              ; point to 1st char in token
        push    bc              ; save token number
        ld      b,a             ; count into b
tok2:   ld      a,(de)          ; get input character
        cp      (hl)            ; compare with token character
        jr      nz,nxttoken     ; if no match forget this token
        inc     hl              ; point to next character
        inc     de
        dec     b               ; reduce remaining chars count
        jr      nz,chkchars     ; if b isn't 0 then jump
        ld      a,c
        ld      (len),a         ; store it away
        pop     bc              ; token found - restore token no.
        pop     hl              ; clear stack
        pop     hl              ;
        pop     hl              ;
        ex      de,hl           ; put new input address into hl
        dec     hl              ; and dec to keep main loop in step
        xor     a               ; set zero flag to signal success
        ld      a,b             ; get token number into a
        ret
chkchars:
        dec     c               ; reduce remaining character count
        jr      nz,tok2         ; if still chars to check go back
nxttoken:
        pop     bc              ; restore remaining chars+tok no.
        dec     b               ; next token number
        pop     hl              ; get back pointer to curr tok. no
        ld      a,(hl)
        ld      e,a             ; length into de
        ld      d,0
        inc     hl              ; step over count
        add     hl,de           ; hl now point at next tokens count
        pop     de              ; restore input address
        jr      tok1            ; jump to check next token

notfound:
        pop     hl              ; restore original input address
        ld      a,(hl)          ; restore character
        or      a               ; reset zero flag to signal failure
        ret


        ; subroutine to check for 4 bit characters

fourbit:push    hl              ; save input address
        ld      hl,tab4bit      ; point hl at character table
        call    tabscan         ; scan table for value
        pop     hl              ; restore input address
        ret                     ; go back

        ; subroutine to check for 8 bit characters
eightbit:
        push    hl              ; save input address
        ld      hl,tab8bit      ; point at 8 bit character table
        call    tabscan         ; scan table
        pop     hl              ; restore address
        ret                     ; return

        ; general purpose table scanning routine

tabscan:ld      b,0ffh          ; 16 characters in each table
                                ; (one is numbered 0)
tabsc2: cp      (hl)            ; check character
        jr      z,tabsc3        ; return with 0 set if match
        inc     hl              ; check next table element
        djnz    tabsc2          ; loop till no chars in table
        or      a               ; reset 0 flag to signal no match
                                ; a contains char code <> 0
        ret                     ; go back
tabsc3: ld      a,b             ; put nibble into a
        ret

        ; routine to check for valid characters
        ; only space  , . ; : and upper case letters allowed
check:  cp      ' '             ; check for space first
        ret     z
        cp      ','             ; and comma
        ret     z
        cp      '.'             ; finally full stop
        ret     z
        cp      'A'             ; assume contiguous alphebet
        jr      c,nogood        ; jump if 'less than' A
        cp      'Z'+1
        jr      nc, nogood
        ld      c,a
        xor     a               ; set zero to signal character ok
        ld      a,c
        ret
nogood: ld      a,255
        and     a               ; reset zero to signal failure
        ret

writenib:
        ld      c,a             ; save nibble
        ld      a,(mask)        ;
        ld      de,(output)     ; get address of byte to write to
        and     a               ; check mask value
        jr      nz,left         ; jump if nibble needs shifting
        ld      a,(de)          ; get old nibble
        or      c               ; insert new nibble
        ld      (de),a          ; and store it back
        inc     de              ; point to next byte
        ld      (output),de     ; save address
        ld      a,255
        ld      (mask),a        ; signal left nibble next time
        ret
left:   ld      a,c             ; get value into a
        sla     a                       ; shift it across
        sla     a
        sla     a
        sla     a
        ld      (de),a          ; and store it
        xor     a
        ld      (mask),a        ; signal other nibble next time
```

```
        ret

; 4 Bit text expansion program for Z80, machine independent

expand: ld      hl,(string)     ; string contains address of
                                ; string to expand
        inc     hl              ; point hl at start of string
        ld      (string),hl     ; put back new string start
        ld      de,(output)     ; point at start of output space
        push    de              ; save for end
        ld      a,255
        ld      (mask),a        ; handle left nibble first

nextnib:call getnib             ; get next input nibble
        cp      2               ; is this the end
        jp      z,fin           ; if so jump to finish up
        jp      nc,exp4bit      ; if nibble ) 2 process 4 bits
        and     a
        jp      z,exp8bit       ; if nibble=1 process 8 bit char
        call    getnib          ; get nibble to expand into token
        call    tokscan         ; find the one we want
tokloop:ld      b,a             ; count into b
tklp1:  ld      a,(hl)          ; get token character
        inc     hl              ; point to next character
        call    outchar         ; output to new string
        djnz    tklp1           ; loop for all the token chars
        jr      nextnib         ; jump back for next input nibble
tokscan:ld      hl,tktable
        ld      b,a             ; initialise b
        ld      a,0fh
        sub     b
        ld      b,a             ; count into b
        inc     b
toksc1: ld      a,(hl)          ; get lenght of this token
        inc     hl              ; point at first char in token
        dec     b               ; reduce count
        ret     z               ; return if it's the one we want
        ld      e,a             ; step over this token
        ld      d,0
        add     hl,de
        jr      toksc1          ; jump back

fin:    ld      hl,(output)     ; get end of output string in hl
        pop     de              ; start of the output string
        and     a               ; clear carry
        sbc     hl,de           ; length in hl
        ld      a,l             ; length mustn't be ) 255
        ld      (de),a          ; put count in right place
        ret                     ; return to system

exp8bit:call    getnib          ; get next nibble for 8 bit char
        ld      hl,tab8bit      ; point hl at correct table
        call    index           ; get the right character
        call    outchar         ; put it in the output string
        jr      nextnib         ; go back for more

exp4bit:ld      hl,tab4bit      ; point at 4 bit table
        call    index           ; find character
        call    outchar         ; output it
        jr      nextnib         ; round again

index:  ld      e,a             ; initialise e
        ld      a,0fh
        sub     e               ; get offset into a
        ld      e,a             ; put into e
        ld      d,0
        add     hl,de           ; index into table
        ld      a,(hl)          ; get character
        ret

outchar:ld      de,(output)     ; get last address outputted to
        inc     de              ; point to new address
        ld      (de),a          ; store character
        ld      (output),de     ; put back new address
        ret

getnib: ld      a,(mask)        ; get mask value
        ld      hl,(string)     ; get address of input byte
        and     a
        ld      a,(hl)          ; get byte
        jr      nz,shfta        ; jump if left nibble required
        and     0fh             ; mask right nibble
        inc     hl              ; point to new byte
        ld      (string),hl     ; store for next time
        ld      c,a
        ld      a,255           ; signal left nibble next time
        ld      (mask),a
        ld      a,c
        ret
shfta:  sra     a               ; shift right nibble ove
        sra     a
        sra     a
        sra     a
        and     0fh             ; mask it off
        ld      c,a
        xor     a               ; signal right nibble next time
        ld      (mask),a
        ld      a,c
        ret

        ; table of values

        ; four bit table

tab4bit:db      ' '             ; space character
        db      'E'
        db      'T'
        db      'A'
        db      'O'
        db      'N'
        db      'R'
        db      'I'
        db      'S'
        db      'H'
        db      'D'
        db      'L'
        db      'F'
        db      0               ; dummy values to fill table
        db      0
        db      0

        ; 8 bit characters

tab8bit:db      'C'
        db      'M'
        db      'U'
        db      'G'
        db      'Y'
        db      'P'
        db      'W'
        db      'B'
        db      'V'
        db      'K'
        db      'X'
        db      'J'
        db      'Q'
        db      'Z'
        db      '.'
        db      '*'
        ; token table
tktable:db      3,'THE'
        db      4,'THIS'
        db      4,'THAT'
        db      2,'IF'
        db      3,'YOU'
        db      2,'ME'
        db      3,'WAS'
        db      2,'HE'
        db      3,'SHE'
        db      4,'THEY'
        db      2,'OF'
        db      2,'IT'
        db      2,'IS'
        db      3,'FOR'
        db      2,'ON'
        db      2,'TO'
        db      0               ; end of table marker

        end
```

Die Qual der Wahl

Für die meisten Hobby-Anwender reicht BASIC zur Programmentwicklung völlig aus. In Sonderfällen braucht man jedoch besser angepaßte Sprachen. Eine gründliche Analyse des Vorhabens sollte die Qual der Wahl jedoch lindern können.

Bis vor kurzem stand dem Heimcomputer-Freund ausschließlich BASIC oder Maschinensprache zur Verfügung. Heute sind dagegen fast alle verbreiteten Sprachen auch für die kleinsten Rechner erhältlich, die gesamte Palette steht beim IBM-Kompatiblen oder anderen MS-DOS-Rechnern zur Verfügung. Compiler und Interpreter können recht teuer sein, daher arbeiten nur wenige Anwender mit mehr als drei Sprachen.

Effektivität in bezug auf Speichergröße und die Arbeitsgeschwindigkeit sind die wichtigsten Faktoren. Das gilt besonders bei Programmen, die mit der Außenwelt kommunizieren sollen. Manche Programme wären in BASIC so schwierig zu erstellen, daß es leichter ist, eine neue, besser geeignete Sprache zu lernen und das Programm damit zu entwickeln. Ein Pro-

Sprache: BASIC
Vorteile: Leicht erlernbar und einfach in der Anwendung, weit verbreitet. Preiswert. Vielseitige Arithmetik-Funktionen. Komfortable Stringverarbeitung.
Nachteile: In vielen Versionen kaum Möglichkeiten für Modulprogrammierung und Steuerstrukturen. Keine Norm. Programmausführung langsam. Schlechtes File-Handling. Eingeschränkte Zahl von Datentypen und -strukturen. Programme undurchsichtig.
Anwendung: Kurze, einfache Programme mit Berechnungen oder Stringverarbeitung.

Sprache: LOGO
Vorteile: Solide mathematische Grundlage, leicht zu lernen. Turtle-Grafik. Gute Möglichkeiten zur Listenverarbeitung und für modulares Programmieren, viele Datentypen und Strukturen. Preiswert.
Nachteile: Auf höherer Ebene ist die Programmierung schwierig. Viele nicht-kompatible Versionen. Ausführung langsam.
Anwendung: Grafik, Listenverarbeitung, Lernen von Mathematik und Einführung in höhere Programmkonzepte.

Sprache: PASCAL
Vorteile: Gut strukturiert. Viele Datentypen. Weitgehend standardisiert. Verbreitet. Einfache Programmierung.
Nachteile: Ein/Ausgabe nicht klar definiert, File-Handling unbefriedigend.
Anwendung: Einüben guter Programmiertechnik. Allgemeine Aufgaben von beschränktem Umfang.

Sprache: FORTH
Vorteile: Schnelle Verarbeitung, vom Anwen-

gramm zur on-line-Verarbeitung von Lagerdateien wäre in COBOL durch den Einsatz indexierter Dateien sehr einfach zu erstellen. In BASIC müßte dagegen mit sequentiellen Dateien oder einem Hashing-Algorithmus und Random Access Files gearbeitet werden.

Bedienerfreundliche, leicht erlernbare Sprachen bereiten bei größerem Programmumfang oft Probleme. Hier kommt es darauf an, daß sich das Programm in selbständige Module zerlegen läßt, die unabhängig voneinander programmiert werden können.

Einige Sprachen, etwa COBOL und FORTRAN, sind gemäß allgemeingültiger Richtlinien fest definiert; die Anpassung an neue Hardware- und Verarbeitungstechniken ist dann recht mühsam und langwierig. Dafür erfordert das auf einem Rechner erstellte Programm nur eine erneute Compilierung und nur wenige Änderungen, bis es auf dem anderen Computer läuft.

Sprachen wie PASCAL und C haben nur eine Art de facto-Standard, der von den Erfindern festgelegt wurde. Die meisten Versionen dieser Sprachen entsprechen diesem Standard zwar, bei PASCAL sind jedoch Ein- und Ausgabeprozeduren nicht fest definiert; der Anwender hat also freie Bahn für Zusätze und Änderungen. Die entstandenen Programme sind natürlich nicht problemlos übertragbar.

Ein weiterer Faktor ist das Übersetzungsprogramm. Interpreter sind im allgemeinen einfacher zu bedienen und eignen sich besser für die Programmentwicklung, dafür sind sie aber auch langsamer beim Programmablauf. Compiler dagegen sind schwierig anzuwenden, das Endprodukt arbeitet dafür effektiver. Die Unterschiede nehmen allerdings durch die Entwicklung von Debugger-Programmen und komfortablen Compilern zunehmend ab.

Einige Sprachen arbeiten durch spezielle Hardware-Neuerungen sehr viel schneller. So wurde viel Aufwand in die Entwicklung eines Prozessors gesteckt, der direkt in FORTH arbeiten kann.

Der Hauptfaktor bleibt aber immer die Anwendung, für die eine Computersprache eingesetzt werden soll. Wir möchten Ihnen hier die wichtigsten Sprachen mit ihren Vorzügen und Schwachpunkten vorstellen.

der auszubauen. Gute Strukturierung.
Nachteile: Das Niveau ist oft zu niedrig. Schwer verständlich. Viele verschiedene Versionen.
Anwendung: Programmierung auf maschinennahem Niveau, speziell bei der Hardwaresteuerung.

Sprache: FORTRAN
Vorteile: Standardisiert. Umfangreiche Software-Bibliothek. Sehr viele Funktionen.
Nachteile: Altmodisch. Unbefriedigende Strukturierung. Programme oft unverständlich. Festgelegte Programmgestaltung.
Anwendung: Allgemeine wissenschaftliche, mathematische Programme und Anwendungen aus dem Produktionsbereich.

Sprache: COBOL
Vorteile: Standardisiert, für viele Rechner. Leicht verständlich. Gutes File-Handling.
Nachteile: Schwer zu erlernen. Compiler sind umfangreich und teuer. Schlecht strukturiert, geringe Zahl von Datentypen.
Anwendung: Kommerzielle Datenverarbeitung.

Sprache: C
Vorteile: Viele Datentypen und Strukturen. Gute modulare Struktur. Einfacher Zugang zur Hardware. Programme sind schnell.
Nachteile: Programme unübersichtlich. Für ernsthafte Anwendungen ist das Niveau zu niedrig. Nicht völlig standardisiert.
Anwendung: Systemsoftware als Ersatz für Assembler. Für besonders schnelle Anwendungen.
Sprache: LISP
Vorteile: Gute Listenverarbeitung, vielseitige mathematische Funktionen. Umfangreiche Software und Hilfsprogramme. Bei größeren Rechnern weit verbreitet.
Nachteile: Schwer erlernbar, schwer anzuwenden.
Anwendung: Künstliche Intelligenz und allgemeine Listenverarbeitung.

Sprache: PROLOG
Vorteile: Gute mathematische Basis. Läßt sich einfach einsetzen. Steht dem menschlichen Denken nahe.
Nachteile: Keine rein relationale Sprache, es gibt verfahrensorientierte Ansätze.
Anwendung: Künstliche Intelligenz, Datenbanksysteme.

Sprache: Assembler
Vorteile: Volle Kontrolle über alle Rechnerkomponenten. Effektivste Programmiersprache.
Nachteile: Keinerlei Norm. Schwer zu erlernen und anzuwenden.
Anwendung: Höchste Effizienz, aber nur, wenn sonst „nichts geht".

Sprachgefühl

In der Tabelle ist eine Reihe von Anwendungen aufgeführt. Die passenden Sprachen stehen nach Eignung geordnet rechts daneben.

Anwendung	Sprachen
Statistische Analysen	FORTRAN, BASIC, PASCAL
Lagerverwaltung	COBOL, PASCAL, BASIC
Robotersteuerung	FORTH, C, Assembler
Adventure-Spiele	PROLOG, C, BASIC
Expertensysteme	LISP, PROLOG, LOGO
Ausbildung	LOGO, PROLOG, PASCAL

Digitaler Dialog

Über die Ein- und Ausgabe läuft die Kommunikation des Microprozessors mit den Speichereinheiten und der Peripherie. Über sogenannte Adreßleitungen können mehr als 65 000 Speicherplätze und 256 Peripheriegeräte angesprochen werden.

Ein- und Ausgabe, oder die Abkürzung ‚E/A' (Input/Output – I/O) bezeichnet die Übermittlung von Informationen zwischen der CPU und der „Außenwelt". „Außenwelt" sind in diesem Zusammenhang Peripheriegeräte, die an den Computer angeschlossen sind. RAM- und ROM-Speicher sind in diese Bezeichnung nicht mit eingeschlossen, da sie als fest integrierter Bestandteil des Computers gelten. Ansonsten aber ist die Unterscheidung, was nun ‚innerhalb' und was ‚außerhalb' eines Computers existiert, nicht klar definiert.

Es gibt eine Vielfalt von Peripheriegeräten, die E/A für die Kommunikation mit dem Computer verwenden, darunter fallen Tastaturen, Diskettenlaufwerke, Joysticks, Drucker und Bildschirme.

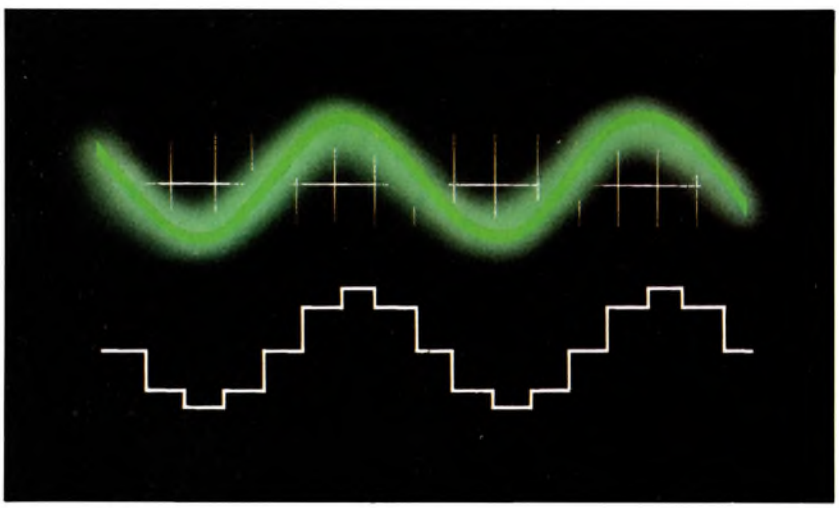
Doch wie funktioniert nun E/A? Will die CPU Daten aus einem Speicher lesen, muß sie zunächst die „Adresse" der Speicherstelle ansprechen, auf der die Daten gespeichert sind. Im umgekehrten Fall – wenn die CPU Daten zum späteren Gebrauch speichern will, muß sie ebenfalls als erstes die Adresse der Speicherstelle ansprechen, auf der die Daten gespeichert werden sollen. Dieser Vorgang wird „Speicheradressierung" genannt. Dabei gibt die CPU die binär codierte Speicheradresse über eine Gruppe von 16 Leitungen aus, die mit den Adreß-Kontakten der CPU verbunden sind. Diese 16 Leitungen werden „Adreß-Leitungen" oder „Adreßbus" genannt.

Speicheradressierung

Spezialschaltungen in der Speicherverwaltung entschlüsseln die 16 Bits und wählen die entsprechende Speicherstelle an. (16 Bits oder Binärstellen haben die Möglichkeit, 65 536 Kombinationen von Nullen und Einsen zu bilden, und können daher ebenso viele Speicherstellen adressieren.)

Soll der Computer mit einem externen Gerät Verbindung aufnehmen, muß dieses in ähnlicher Weise lokalisiert werden. Dafür stehen jedoch nur acht Adreßleitungen zur Verfügung, so daß sich die Summe der E/A-Speicherplätze auf 256 reduziert. Im Vergleich mit den Adressiermöglichkeiten der 16 Adreßleitungen ist diese Zahl sehr klein, in der Praxis reicht sie aber völlig aus, da normalerweise keine Notwendigkeit besteht, eine noch größere Anzahl externer Geräte an einen einzelnen Computer anzuschließen.

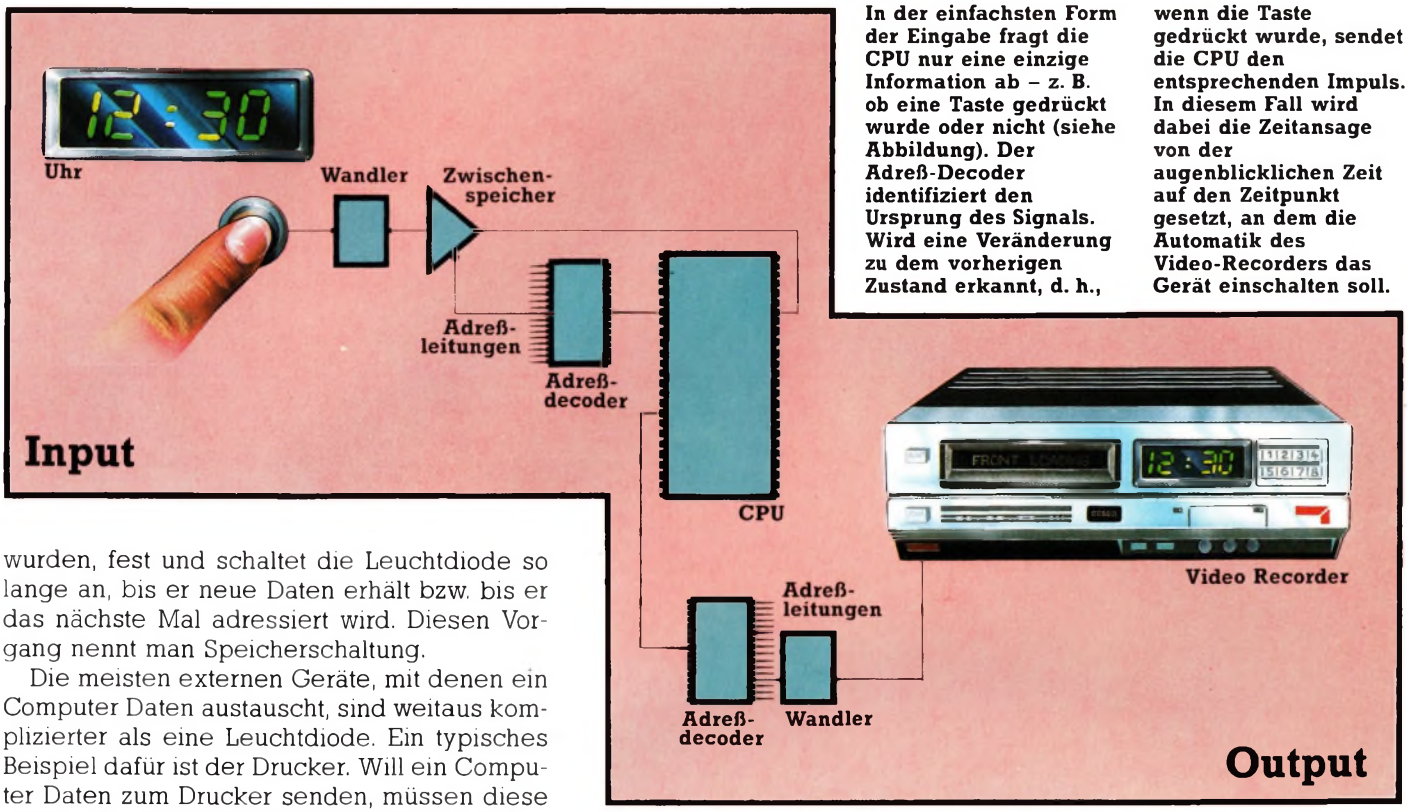
Als Beispiel dafür, wie ein Computer ein Peripheriegerät anspricht, nehmen wir einen der einfachsten Ausgabemechanismen, die es gibt: Eine in die Tastatur eingebaute LED-Anzeige (Light Emitting Diode – Leuchtdiode), die aufleuchtet, wenn die ‚caps lock'-Taste gedrückt, also auf Großschreibung umgestellt ist. Für den Computer ist die LED-Anzeige ein Peripheriegerät, dem er Daten senden kann. Im Fall einer einzigen Leuchtdiode bestehen die Daten zwar nur aus einer 1 (LED – AN) oder einer 0 (LED – AUS), aber – obwohl die Steuerung mit einem einzigen Bit erfolgt, muß dafür eine Adresse zur Verfügung stehen. Nun kann sich die CPU nicht die ganze Zeit damit beschäftigen, eine einzige LED zu steuern. Es muß also eine Möglichkeit geben, die LED nur einmal anzusprechen, um ihr mitzuteilen, daß sie angeschaltet oder daß sie abgeschaltet werden soll.



Mark Watkinson

Die reale Welt enthält wenig Daten oder Informationen, die genau abgegrenzte, digitale Form haben. Daten treten meist in Wellen auf, z. B. die Gezeiten oder Geräuschpegel. Soll ein Computer diese Art von Daten verarbeiten, müssen sie in digitale Form umgesetzt werden. Ein Analog/Digital-Wandler (A/D) nimmt in bestimmten, festgesetzten Zeitabständen – etwa hundertmal pro Sekunde – Werte von der Signalquelle auf. Jeder dieser Werte wird in einer separaten Speicherstelle in digitaler Form festgehalten und ermöglicht damit die Durchführung von Rechenoperationen, z. B. zur Varianzanalyse oder zur Erkennung von Werten, die bestimmte gesetzte Grenzen überschreiten. Der Digital/Analog-Wandler (D/A) führt den gleichen Vorgang in umgekehrter Richtung aus. Dabei werden statistische Techniken angewendet, um die einzelnen Impulse in eine glatte Kurve umzusetzen.

Nehmen wir an, die LED hat die Speicheradresse 32 – in binärer Codierung 00100000. Der spezielle Decoder der LED ignoriert nun alle Bitkombinationen des Adreßbusses außer der Zahl 00100000. Wird diese Zahl ausgegeben, gibt wiederum der Decoder eine hohe Spannung und damit das Signal „wahr" aus. Ein Spezialchip, der ‚data latch' (Zwischenspeicher) genannt wird, übernimmt jetzt das Signal. Er hält die Daten, die ihm gesendet



In der einfachsten Form der Eingabe fragt die CPU nur eine einzige Information ab – z. B. ob eine Taste gedrückt wurde oder nicht (siehe Abbildung). Der Adreß-Decoder identifiziert den Ursprung des Signals. Wird eine Veränderung zu dem vorherigen Zustand erkannt, d. h., wenn die Taste gedrückt wurde, sendet die CPU den entsprechenden Impuls. In diesem Fall wird dabei die Zeitansage von der augenblicklichen Zeit auf den Zeitpunkt gesetzt, an dem die Automatik des Video-Recorders das Gerät einschalten soll.

wurden, fest und schaltet die Leuchtdiode so lange an, bis er neue Daten erhält bzw. bis er das nächste Mal adressiert wird. Diesen Vorgang nennt man Speicherschaltung.

Die meisten externen Geräte, mit denen ein Computer Daten austauscht, sind weitaus komplizierter als eine Leuchtdiode. Ein typisches Beispiel dafür ist der Drucker. Will ein Computer Daten zum Drucker senden, müssen diese Daten die vollständige Codierung für jeden einzelnen Buchstaben, der gedruckt werden soll, enthalten. Sind große Mengen an Informationen zu übermitteln, wird dafür ein spezieller E/A-Schnittstellen-Chip eingesetzt. Diese Art von Chips vereinfacht die Konstruktion eines Computers sehr, da ein Großteil der Schnittstellenschaltung auf dem Chip bereits vorhanden ist. Ein weitverbreiteter Chip dieser Art ist der 8255 PPI (Programmable Peripheral Interface – Programmierbare Peripherieschnittstelle). Der Chip enthält 40 Kontakte, die drei Ein- und Ausgänge im 8-Bit-Modus steuern können, d. h., es stehen 24 Kontakte für E/A zur Verfügung – acht für jedes Peripheriegerät. Jede dieser Achtergruppen kann acht Bits (ein Byte) auf einmal z. B. an einen Drucker senden oder auch acht Bits von einem Eingabegerät, wie z. B. einer Tastatur, empfangen.

Zwischenspeicherung

Um acht Datenbits an einen Drucker senden zu können, adressiert die CPU zunächst die PPI und übermittelt dann acht Datenbits über den Datenbus. Die Daten werden innerhalb des Chips in einem temporären Speicher mit einem Byte Kapazität (Register) zwischengespeichert und dann von der PPI an die entsprechende Gruppe von E/A-Kontakten gesandt. Die Eingabe von Daten in den Computer funktioniert in umgekehrter Weise: Die externen Daten werden Byte für Byte im internen Register der PPI gespeichert und erst dann über den Datenbus gesendet, wenn die CPU das entsprechende Signal dazu gibt. Externe Geräte dürfen mit ihren Daten nicht ständig den Datenbus des Computers belegen, da dieser für die Übertragung von internen Daten und auch für den Empfang von Daten anderer Peripheriegeräte benötigt wird. Deshalb speichert der E/A-Chip die Daten kurzzeitig und gibt sie nur dann über den Datenbus aus, wenn die CPU den Befehl dazu gibt.

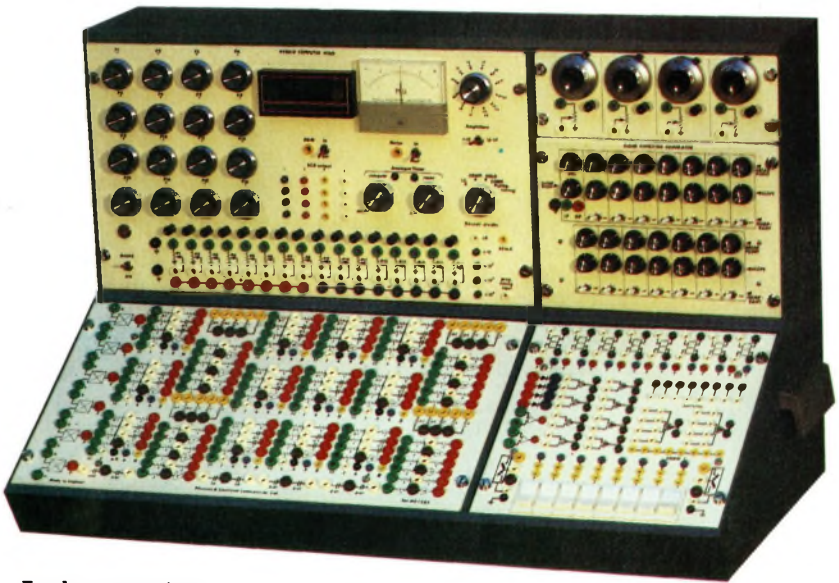
Wodurch merkt nun die CPU, daß ein Peripheriegerät versucht, Daten an den Computer zu senden? Für diesen Vorgang gibt es zwei unterschiedliche Methoden. Bei der ersten unterbricht die CPU das laufende Programm in bestimmten Zeitabständen und überprüft alle Eingabekanäle. Sind dort eingehende Daten vorhanden, wird die Schnittstelle instruiert, die Daten auf den Datenbus auszugeben.

Die andere Methode verwendet sogenannte ‚interrupts' – Unterbrecherschaltungen. Dabei sendet ein Peripheriegerät, das Daten übermitteln möchte, ein Unterbrechersignal direkt an die CPU, hält damit das laufende Programm an und übermittelt seine Daten.

Die oben beschriebene E/A-Methode wird ‚parallele E/A' genannt, da dabei ein vollständiges Byte über acht gleichzeitig aktive Leitungen übertragen wird. Eine andere Methode ist die ‚serielle E/A'. Hierbei wird jedes Bit nacheinander und einzeln übermittelt. Es gibt eine große Anzahl von Druckern, die serielle Schnittstellen verwenden. Auch die Datenübermittlung von Modems funktioniert im seriellen Modus. Der größte Vorteil der seriellen Kommunikation liegt darin, daß nur zwei Leitungen nötig sind, während für die parallele acht Leitungen gebraucht werden.

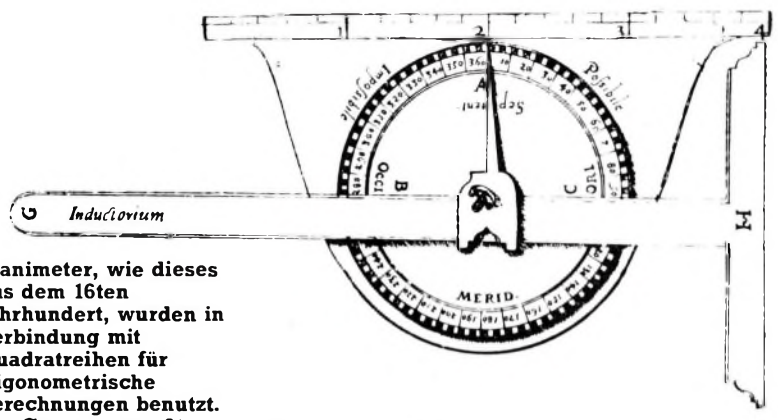
Analoge Systeme

Im Gegensatz zu digitalen Rechnern reagieren analoge Computer direkt proportional auf Veränderungen, ohne eine Umsetzung in digitale Werte zu benötigen.



Analogcomputer benutzen keine Sprachen wie zum Beispiel BASIC. Sie werden „programmiert", indem man die einzelnen elektrischen Einheiten, die sich auf der Rückseite befinden, mit Kabeln verbindet.

Nicht alle Computersysteme verarbeiten Daten digital, also über Binärziffern (digits). Es gibt auch Systeme, die Daten und Informationen analog umsetzen. Der Tachometer beispielsweise ist ein einfacher analoger Computer, denn die Geschwindigkeit des Fahrzeuges ist direkt proportional zur Position



Planimeter, wie dieses aus dem 16ten Jahrhundert, wurden in Verbindung mit Quadratreihen für trigonometrische Berechnungen benutzt. Die Geometer mußten Messungen durchführen und später mathematische Berechnungen, um dadurch die lange Dreiecksseite zu bestimmen. Dieses Analoginstrument vereinfachte die Arbeit ganz erheblich, und führte zu exakteren Ergebnissen.

der Tachonadel.

In der Entwicklung der Elektronik entdeckte man schon früh die Parallelität zwischen elektronischen Bauelementen, wie zum Beispiel einer Spule, und mechanisch arbeitenden Geräten. So fanden Ingenieure heraus, daß die Schwingungen in einem Stromkreis, der aus einer Spule und einem Kondensator besteht, den gleichen Gesetzen unterliegen wie die Schwingungen eines Körpers, der an einer Stahlfeder hängt. Beide physikalischen Vorgänge haben ein und dieselbe mathematische Grundlage – die Basis für die Entwicklung analoger Computer war geschaffen.

Moderne analoge Computer können die vielfältigsten Aufgaben übernehmen. Ihre elektronischen Bauelemente, wie Transistoren, Kondensatoren oder Widerstände, findet man in vielen Haushaltsgeräten wieder. Einige dieser Analoggeräte sind den Originalen fast wirklichkeitsgetreu nachgebildet. So zum Beispiel Flugzeugmodelle für Windkanaltests. Diese Modelle sind exakte, verkleinerte Kopien des Flugzeugrumpfes. Es gibt aber auch analoge Darstellungen, die völlig abstrakt erscheinen. Die Darstellung einer bestimmten Lebenssituation kann – auf analoge Weise – in einer Liste mathematischer Formeln umgesetzt sein, und ein Schaltkreis kann beispielsweise den Wasserfluß eines Stauwehrs symbolisieren.

In der Praxis wurden analoge Prinzipien bereits Anfang des 17. Jahrhunderts angewendet, und zwar bei dem 1630 von William Oughtred entwickelten Rechenschieber. Beim Rechenschieber werden Zahlen auf zwei beweglichen Schienen so verteilt, daß bei Verschiebung der Schienen multipliziert werden kann. Das Ergebnis kann dann direkt auf einer der Schienen abgelesen werden. Ende des 19. Jahrhunderts entwickelte Lord Kelvin eine Handkurbelmaschine, mit der man den Tidenhub, also die Wasserstandsveränderung von Ebbe zu Flut, in einem Hafen berechnen konnte.

Addieren durch Stromstärke

Der erste elektronisch gesteuerte analoge Computer wurde 1947, kurz nach der Entwicklung der digitalen Rechner, in Betrieb genommen. Wie bereits erläutert, führen Digitalrechner Rechenoperationen mit Hilfe sogenannter logischer Gatter durch. Analoge Computer hingegen nutzen bei ihren Kalkulationen die speziellen Eigenschaften der Elektrizität aus. Dazu ein Beispiel: Durch einen Draht fließt Strom von fünf Ampere, durch einen zweiten Draht Strom von vier Ampere. Werden nun beide Drähte miteinander verbunden, dann beträgt die Stromstärke neun Ampere, also genau die Summe aus beiden. Allein durch die Eigenschaften des Stroms entsteht ein „Addierer".

So lassen sich oft schwierige mathematische

Probleme durch einfach aufgebaute Schaltkreise lösen. Ein Beispiel hierfür ist die Integralrechnung, die Berechnung einer Fläche unter einer Kurve. Da analoge Computer anders „programmiert" sind als digitale Rechner, muß zur Lösung des Problems ein Schaltkreis konstruiert werden, der das Problem auf elektronische Weise repräsentiert. Dazu werden alle Bauelemente hinter ein Steckbrett montiert, auf dessen Vorderseite die Verbindungen zwischen den einzelnen Bauteilen über Stekker und Kabel hergestellt werden. Eine solche Anlage ähnelt sehr stark einem altmodischen Telefonschaltbrett.

In einem analogen Computer werden die veränderlichen Ströme und Spannungen zur Darstellung verschiedener realer physikalischer Größen, wie beispielsweise Kräfte oder Geschwindigkeiten genutzt. In Digitalrechnern dagegen werden alle Daten durch eine Reihe elektronischer Impulse dargestellt, die binäre Eins durch fünf Volt, die Null durch null Volt. Ein weiterer Unterschied zwischen einem analogen und einem digitalen Computer liegt in der Art der Speicherung der Daten. In Analoggeräten werden Informationen durch sich verändernde Werte gespeichert, in Digitalgeräten durch einzelne, „diskrete" Einheiten.

Fehlerlose Datenübertragung

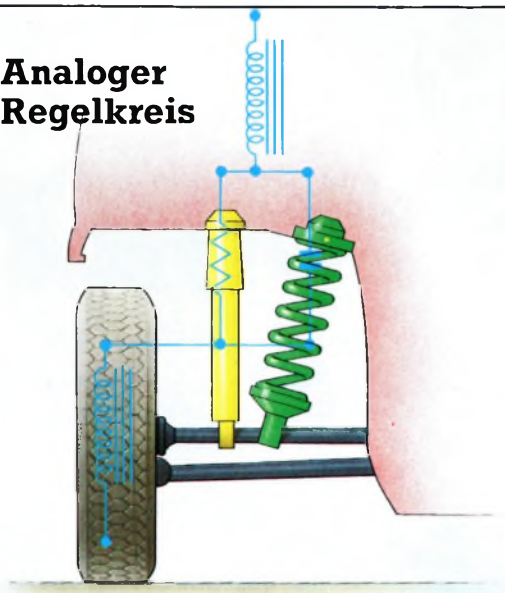
Der große Vorteil der Digitalrechner liegt in ihrer Fähigkeit, Daten ohne Verluste und fehlerlos übertragen und bearbeiten zu können. Wird ein Stromstoß von zehn Volt durch einen Schaltkreis geschickt, so kann er durch Materialeigenschaften abgeschwächt werden und dann nur noch 9,9 Volt betragen. In einem analogen System, in dem Schwankungen der Stromspannung eine Änderung der Information bedeuten, würde der Verlust von 0,1 Volt zu einer neuen Information führen. Stellen beispielsweise zehn Volt den Ton A einer Singstimme dar, so verändert sich der Ton bei einem Spannungsabfall auf 9,9 Volt unter Umständen schon auf Gis.

In einem digitalen System hingegen gibt es nur zwei mögliche Signale, entweder 5 oder 0 Volt, und jede Spannung in der Nähe von 5 Volt wird automatisch als 5 Volt erkannt und genau auf diesen Wert gebracht. So werden Abweichungen korrigiert und Fehler können sich nicht gegenseitig verstärken. Dies ist der Nachteil der analogen Computer, daß Signale beim Durchlaufen mehrerer Schaltkreise immer mehr verfälscht werden können und der Fehler dadurch zunehmend verstärkt wird.

Trotzdem erzielen analoge Computer einen Vorteil in der Darstellung von Größen durch verschiedene Stromstärken oder -spannungen. Denn das System reagiert sofort, sowie sich die Eingangswerte ändern. Dies gilt auch für sehr schnelle Änderungen, da keine Zeitverluste durch die Codierung der Größen in digitale Werte, deren Bearbeitung und die Decodierung für die Ausgabe auftreten. Diese Eigenschaft ist für Anwendungen, die schnellste Reaktionen erfordern, unentbehrlich. Beispielsweise muß ein Autopilot im Verkehrsflugzeug beim Landeanflug auf eine Windböe mit einer Geschwindigkeit reagieren, die über längere Berechnungen selbst eines modernen Digitalcomputers nicht zu erzielen ist. Sensoren registrieren die Windböe und generieren eine kleine Spannung. Der Schaltkreis des Autopiloten reagiert unverzüglich mit einer relativ größeren Änderung der Ausgangsspannung, die automatisch die Querruder steuert, um das Flugzeug in seiner alten Fluglage zu halten.

Aber auch wenn die analogen Rechner weiterhin Anwendung finden, so werden sie doch durch die technische Fortentwicklung der digitalen Systeme, vor allem hinsichtlich einer größeren Verarbeitungsgeschwindigkeit, immer stärker vom Markt verdrängt werden.

Analoger Regelkreis



Stoßdämpfer bestehen aus einer Feder und einem Dämpfer. Die Feder absorbiert plötzliche Stöße und die entstehenden Schwingungen werden durch den Dämpfer unterdrückt. Ingenieure müssen das beste Verhältnis von Feder und Dämpfer wählen, um den optimalen Fahrkomfort jederzeit zu gewährleisten. Mit einem elektrischen Schaltkreis kann man das System aus Feder, Dämpfer, Auto und Rad nachahmen. Man simuliert die Stöße durch eine Wechselspannung und verändert die elektrischen Werte, bis die Ausgangsschwingung so klein wie irgend möglich ist.

Falsch verstanden!

Die Art, in der ein solcher Satz verändert wird, ähnelt sehr stark den akkumulierenden Fehlern in einem analogen Schaltkreis.

In einem digitalen System können nur zwei Meldungen übertragen werden: Null und Eins. Falls also eine Veränderung auftritt, kann sie leicht erkannt, und in der nächsten Stufe korrigiert werden.

EPROM-Brenner

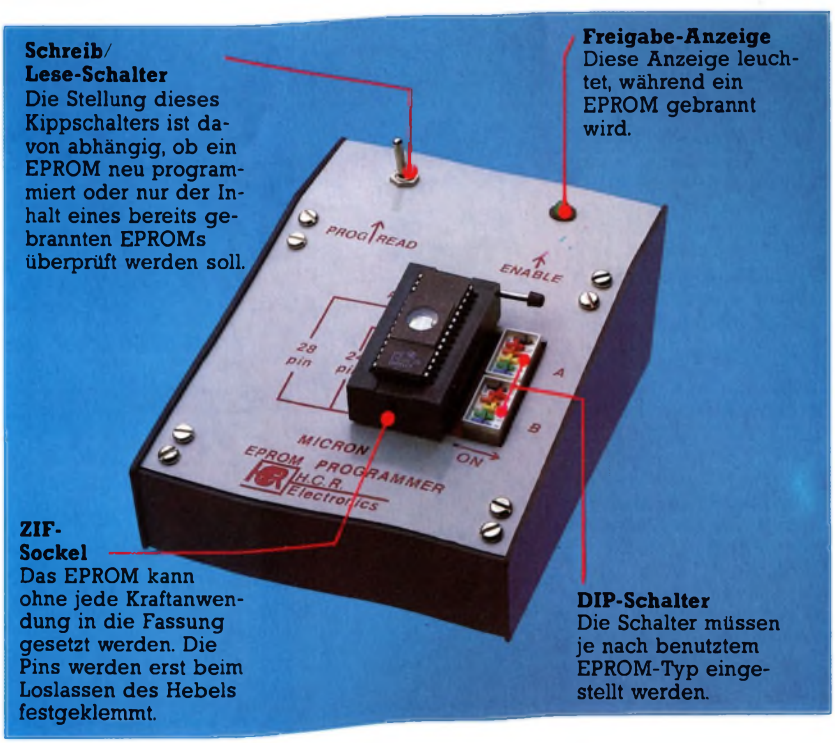
Zur Programmierung eines EPROMs benötigt man einen speziellen EPROM-Brenner. Der hier beschriebene „Micron EPROM Programmer" ist eines der preiswerteren Geräte auf dem Markt.

Durch das Quarzfenster des EPROMs ist der Siliziumchip zu sehen. Auch die Verbindungsdrähte zwischen dem Chip und den Anschlußstiften sind zu erkennen und mit etwas Mühe die Transistorfelder, die die Informationen enthalten.

Mit Hilfe dieses Geräts kann der Benutzer selbstgeschriebene Programme über den User Port des Acorn B in EPROMs „einbrennen", um sie auf den ROM-Steckplätzen des Rechners ständig griffbereit zu halten.

Einige Rechner verfügen über zusätzliche freie ROM-Steckplätze, die der Benutzer für spezielle Zwecke verwenden kann. ROMs mit Anwender-Programmen gibt es inzwischen in großer Auswahl bis hin zu Textverarbeitungs- und Datenbanksystemen oder Sprachen wie LOGO. Der Benutzer kann aber auch selbstgeschriebene Programme in ein EPROM (Erasable Programmable Read Only Memory = lösch- und programmierbarer Festwertspeicher) übertragen und dann in den Rechner einsetzen. Das EPROM wird mit einem speziellen Programmiergerät „gebrannt", das allgemein als „EPROM-Brenner" bezeichnet wird.

Wie alle ROMs besteht ein EPROM aus einer Leiterbahnen-Matrix, wobei an sämtlichen Zeilen/Spalten-Kreuzungen MOS-Speichertransistoren eingebaut sind. Ist ein Speichertransistor leitend programmiert, wird beim Anlegen eines Signals an die Zeile (Wortleitung) eine kreuzende Spalte (Bitleitung) spannungführend (logisch „1"). – Ist der Transistor gesperrt, bleibt die Spalte auf „0". Wenn die CPU über den Adreßbus ein Speicherbyte anfordert, wird auf diese Weise das Informationsmuster der acht Speichertransistoren, die auf der zugeordneten Leitung liegen, auf acht Spaltenleitungen übertragen und über den Datenbus dem Prozessor mitgeteilt.



Bei der EPROM-Herstellung sind die Speicherzellen im allgemeinen auf „1" gesetzt, das heißt, jede Adresse enthält den Hexadezimalwert FF. Die isoliert in Siliziumdioxid eingebetteten Gate-Elektroden der Speichertransistoren sind ungeladen, so daß alle Wortleitungen auf die Datenleitungen „durchgeschaltet" sind. Beim kurzen Anlegen von Überspannung (typisch sind 25V während 50ms) an einen Speichertransistor kommt es zu einem elektrischen Durchbruch im Siliziumdioxid, wobei sich das Gate so auflädt, daß der Transistor gesperrt wird. Wegen der isolierten Einbettung des Gate kann die Ladung nicht wieder abfließen, in der betreffenden Zelle ist daher dauerhaft eine logische „0" gespeichert. Beim Programmieren des EPROMs wird eine Wortleitung nach der anderen adressiert, zur Markierung der zu sperrenden Transistoren jeweils das gewünschte Bitmuster auf die Datenleitungen gegeben und dann der „Einbrenn"-Impuls angelegt.

Programmieren und Lesen

EPROMs können im Unterschied zu PROMs vom Anwender auch wieder gelöscht werden. Dazu ist auf dem Chip ein Quarzfenster angebracht. Bei UV-Bestrahlung wird die Siliziumdioxid-Isolation der Gates leitend, so daß sie sich entladen können. Damit sind die Speichertransistoren wieder durchgeschaltet und alle Bytes auf #FF gesetzt. Ultraviolettes Licht ist nicht unbedingt erforderlich. Es gibt auch „EEPROMs (Electrically Erasable . . .), die elektrisch gelöscht werden können.

Der „Micron EPROM Programmer" von HCR Electronics ist für EPROMs bis zu 16 KByte Speicherkapazität ausgelegt. Das kleine Gehäuse trägt einen speziellen Wechselsockel (ZIF = Zero Insertion Force), der EPROMs mit 24 oder 28 „Beinen" aufnehmen kann. Daneben sind zwei achtpolige DIP-Schalter plaziert, die je nach EPROM-Typ einzustellen sind. Links oben liegt ein Umschalter für Programmier- oder Lesebetrieb. Dieser Hebel verdient besondere Beachtung, weil Fehlbedienung zum Überschreiben eines frisch gebrannten EPROMs führen kann.

Das Gerät wird über ein Flachbandkabel mit dem User Port des Acorn B verbunden. Programme können mit LOAD und SAVE in einen Bufferbereich gebracht und dann auf ein EPROM übertragen werden. Dazu wird auf

Cassette eine menügesteuerte Software geliefert, natürlich mit der Empfehlung, zuerst hierfür ein EPROM zu brennen.

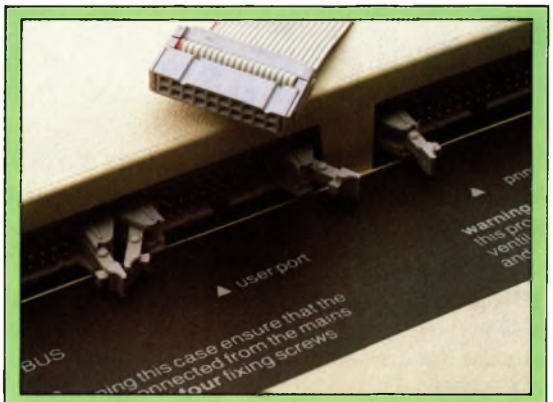
Der Bildschirm zeigt nach dem Laden der Software die anwählbaren Optionen: Vor dem Brennen des EPROMs ist Ihr Programm zunächst mit „L(oad)" in einen ab Byte #2000 reservierten Bufferbereich zu bringen. Anschließend wird der „P(rogram)"-Betrieb aufgerufen. Der Schirm zeigt dann eine Liste aller verwendbaren EPROM-Typen. Wenn Sie gewählt haben, folgt das Einstellschema für die 16 DIP-Schalter, wobei diejenigen blinken, die anders eingestellt werden müssen als beim vorherigen Programm. Zuletzt erscheint die Aufforderung, den Schreib/Lese-Schalter auf „Prog" zu kippen. Danach genügt ein Tastendruck, um des EPROM-Brennen zu starten. Die jeweils bearbeiteten Speicheradressen werden nun fortlaufend angezeigt.

Der Zeitbedarf für den Programmiervorgang hängt davon ab, wie viele Bytes auf #FF bleiben dürfen. Er liegt zwischen 74 Sekunden für ein kurzes Programm und maximal 14 Minuten für ein 16-KByte-EPROM. Mit dem Befehl „V(erify)" können Sie kontrollieren, ob Ihr Programm einwandfrei übertragen worden ist – die Software errechnet eine Prüfsumme und gibt fehlerhafte Adressen auf dem Schirm an. Ist alles in Ordnung, kann das Eprom vorsichtig in einen der vorgesehenen Stecksockel des Computers gesteckt werden.

Sie können für Ihr Programm einen Standard-Kopf festlegen, indem Sie mit dem Befehl „G" einen Namen bestimmen, der später (mit vorgesetztem *) über CALL aufgerufen werden kann. Bei einem BASIC-Programm sollten Sie den „G"-Befehl in Verbindung mit dem Zusatz „F(ill)" verwenden, damit unbenutzte Speicherplätze zur späteren Programmierung auf #FF bleiben. Maschinenprogramme können, auch wenn sie mit Kopf versehen sind, nur ab Adresse #2000 geladen werden.

Auch für Laien geeignet

Angesichts der Nützlichkeit solcher EPROM-Brenner ist es eigentlich erstaunlich, wie wenig sie bei Acorn-Besitzern verbreitet sind. Wahrscheinlich glauben viele, daß diese Geräte in das Reich der begeisterten Elektronikbastler gehören und für „normale" Heimcomputerbenutzer nicht in Frage kommen. Das ist schlicht falsch – so ein EPROM-Brenner ist auch für Laien einfach zu bedienen. Die Verwendung von EPROMs bietet viele Vorteile, wobei am wesentlichsten ist, daß Sie über das Betriebssystem ohne jede Verzögerung auf Ihre Programme zugreifen können, weil das Laden von Diskette oder Cassette entfällt. Außerdem sind die Programme im EPROM von BASIC-Programmen aus aufrufbar. Schließlich werden mit der Bereithaltung der Programme in EPROMs auch viele Störquellen ausgeklammert, die bei der Speicherung auf Magnetträgern die Zuverlässigkeit beeinträchtigen.



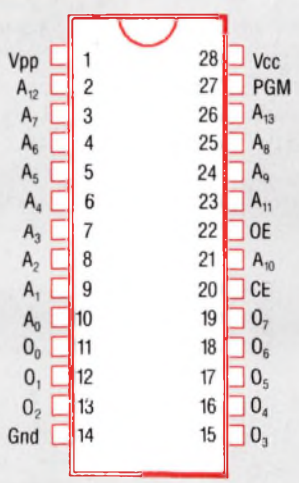
Rechneranschluß
Der EPROM-Brenner wird über ein zwölfadriges Flachbandkabel an den User Port des Acorn B angeschlossen.

EPROM-Brenner gibt es natürlich auch für andere Computer. Die für den Commodore 64 erhältliche Version besteht aus zwei Platinen, die durch ein Flachkabel miteinander verbunden werden sowie einem 2-KByte-EPROM, das die benötigte Software enthält. Die kleinere Platine wird in den Erweiterungssport des C 64 gesteckt. Auf der zweiten Platine befindet sich der eigentliche Programmierer, mit dem 8- und 16-KByte-EPROMs „geschossen" werden können. Die Bedienungsanleitung erklärt, was beim Programmieren zu beachten ist.

Kontaktsuche

Anschlußbelegung	
A_0-A_{13}	Adressen
CE	Chip-Freigabe
OE	Ausgangs-Freig.
O_0-O_7	Ausgänge
PGM	Programmieren
V_{pp} V_{cc}	Stromversorgung
Gnd	Masse (0 V)

Dies ist eine schematische Darstellung des 16-KByte-EPROMs TMS27128 von Texas Instruments mit 28 Anschlüssen, das beim HRC-EPROM-Brenner verwendet werden kann. 14 Pins werden für die Adressierung benötigt, weitere acht für den Datenbus. Mit „V_{cc}" (Pin 28) ist die normale Betriebsspannung, mit „V_{pp}" (Pin 1) die Überspannung für die Programmierung gemeint.



EPROM-Brenner

ABMESSUNGEN
140×105×65 mm

SCHNITTSTELLE
Flachbandkabel-Stekker für den User Port des Acorn B.

DOKUMENTATION
Drei fotokopierte Blätter mit einer Erläuterung der neun Systembefehle, die für Anfänger etwas schwierig sein kann. Dabei ist die Bedienung des Geräts eigentlich sehr einfach.

STÄRKEN
Die Handhabung ist problemlos; die menügesteuerte Software ist vom Benutzer auch ohne viel Verständnis für die technischen Vorgänge beim EPROM-Brennen anwendbar.

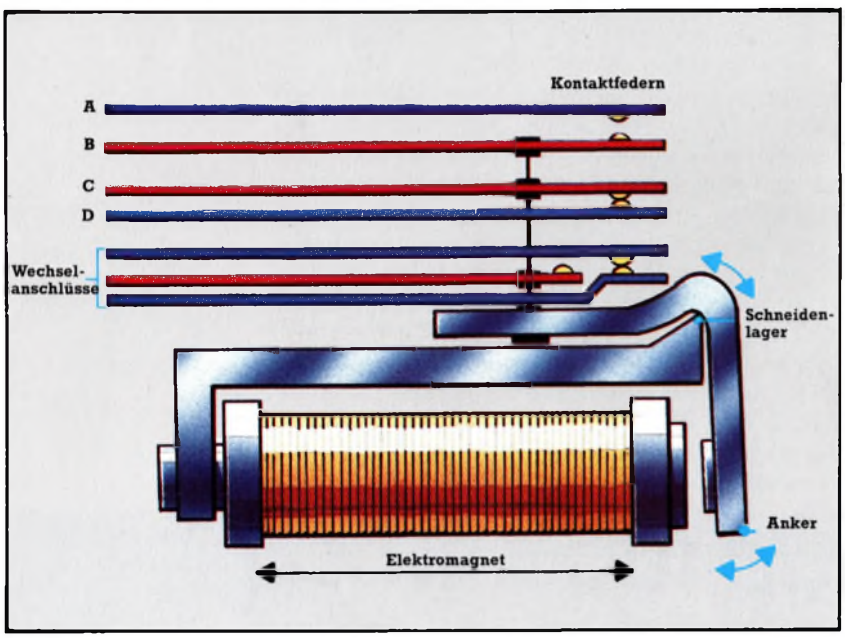
SCHWÄCHEN
Es gibt kaum Hinweise, wie der EPROM-Brenner mittels Erweiterung der Software eingesetzt werden kann. Maschinenprogramme, die nicht auf die Startadresse #2000 verlegt werden können, sind nicht ohne weiteres in ein EPROM zu programmieren.

Stromquelle

Wir setzen den Selbstbau-Kurs mit der Bauanleitung für ein Netzspannungsrelais fort. Damit kann Ihr Computer die Zimmerbeleuchtung zu vorgewählten Zeiten ein- oder ausschalten, aber auch einen Videorecorder sekundengenau steuern.

Elektrische Relais sind Schalter, die durch ein Signal gesteuert werden. In unserem Beispiel sollen große Ströme mit hoher Spannung durch ein Niedervolt-Signal an- und abgeschaltet werden. Relais gibt es in vielen unterschiedlichen Ausführungen, am häufigsten ist aber das Magnetrelais mit Kern, Spule und beweglichem Anker.

Die Relais-Kontakte werden durch Bewegung des Ankers geöffnet und geschlossen. Wenn eine ausreichende Spannung an der um den Weicheisenkern gewickelten Spule liegt,



zieht der Spulenkern den Anker an sich heran: Die am Anker befestigten Kontakte bewegen sich dabei nach oben.

Im Bild ist das Relais in Ruheposition, es ist also keine Spannung an der Spule angelegt. Das Kontaktpaar AB ist dann unterbrochen, das Paar CD aber verbunden. In der Arbeitsposition (wenn Spannung an der Spule liegt) findet man umgekehrte Verhältnisse – A und B sind verbunden, C und D jedoch nicht. Durch diesen Aufbau hat man die Wahl, ob das Steuersignal einen oder zwei Stromkreise öffnen oder schließen soll.

Eine weitere Möglichkeit ist die, das Relais als Wechselschalter einzusetzen. Von den drei untersten Kontakten auf dem Bild sind im Ruhezustand nur der obere und untere Kontakt verbunden. In der Arbeitsposition des Relais drückt die mittlere Kontaktfeder ihr oberes Gegenstück hoch und unterbricht dabei die Verbindung der beiden anderen.

Liste der Bauteile

Anzahl	Bauteil
2	Relais, 250 Volt, 10 A, Steuerspannung 8–15 Volt
2	Kunststoff-Gehäuse mit angespritztem Netzstecker und eingebauter Steckdose
2	Feinsicherungshalter für Platinenmontage
2	Feinsicherungen, 6 Ampere
*	4-mm-Stecker (Bananenstecker)
*	2 Meter 2adriges Flachkabel
*	Lochplatinenreste (Veroboard)
*	Kleine Stücke isoliertes Netzkabel

Die mit * bezeichneten Teile sollten noch von Ihrer letzten Bastelarbeit übrig sein. Mit den angegebenen Teilen können Sie zwei Relaismodule herstellen – der Ausgangsbuffer kann bis zu vier Relais gleichzeitig ansteuern. Wenn Sie kein Fertiggehäuse mit angespritztem Stecker und eingebauter Steckdose bekommen können, läßt sich die Schaltung auch in ein normales Gehäuse einbauen.

Alles im Kasten

Achten Sie bei allen Verbindungen auf Sicherheit und guten Kontakt. Prüfen Sie Ihre Platine genau – zwischen Netz- und Signalkabeln darf es keine Verbindung geben!

Die Relais- und die Sicherungsplatine werden mit Epoxidharz-Kleber im Gehäuse befestigt. Einige Allzweck-Kleber leiten die Elektrizität, diese dürfen keinesfalls verwendet werden! Zur Prüfung sollten Sie etwas Klebstoff auf einem Stück Papier trocknen lassen und den Widerstand mit dem Multimeter messen. Verwenden Sie auch beim kleinsten Ausschlag einen anderen Kleber!

Nach dem Trocknen des Klebstoffes kann das Gehäuse geschlossen werden. Die Bananenstecker an die Signalkabel anschließen. Die Polung ist dabei unwichtig, das Relais arbeitet in jeder Stromrichtung gleich. Die Feinsicherung sorgt dafür, daß das Relais nicht versehentlich überlastet wird. Auch wenn es nach den aufgedruckten Daten bis zu 10 A schalten kann, sollten Sie sich aus Sicherheitsgründen auf 5 A beschränken – also auf Geräte unter 1100 Watt.

Achtung!

*** Vor jeder Arbeit muß der Netzstecker aus der Steckdose gezogen werden!**
*** Prüfen Sie vor dem ersten Einschalten alle Verbindungen sehr genau mit dem Multimeter!**
*** Bauen Sie keine provisorische Schaltung auf – NETZSPANNUNG IST LEBENSGEFÄHRLICH!**

Die Platine

Schneiden Sie die Platine so zu, daß keine Leiterbahn gleichzeitig für die Netz- und die Signalanschlüsse des Relais zuständig ist. Dann das Relais auflöten.

PRÜFEN SIE DIE PLATINE JETZT NOCH EINMAL GENAU! Mit dem Multimeter auf Verbindung zwischen den Bahnen testen – ein Fehler dabei kann tödliche Folgen haben!

Mit isoliertem Netzkabel wird die Verbindung vom einen Pol des in das Gehäuse integrierten Steckers zur Relaisplatine und von dort zu einem Anschluß des Steckdosenteils hergestellt. Der andere Pol wird – ebenfalls durch isoliertes Kabel – mit einem Stück Platine verbunden, auf dem die Feinsicherungshalter festgelötet sind. Nur die Schutzleiteranschlüsse von Stecker- und Steckdosenteil werden mit einer grün-gelben Leitung direkt verbunden. Die Signalleitungen mit den Bananensteckern werden an die Wicklungsanschlüsse des Relais gelötet.

Testprogramm

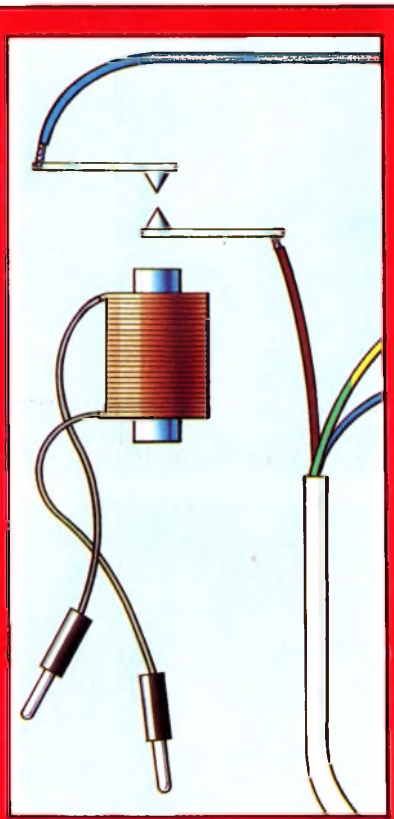
Wenn Ihr Relaismodul fertig und gründlich geprüft ist, können Sie mit diesem Programm testen, ob es die angeschlossenen Geräte richtig ein- und ausschaltet. Nehmen Sie einfach eine kleine Leselampe, und plazieren Sie deren Stecker in die Netzsteckdose des Relaisgehäuses. Die Signalleitungen werden mit dem positiven und negativen Anschluß (Kanal 0) des Niedervolt-Ausgangs verbunden. Jetzt kommt noch der Netzanschluß in die Steckdose.

Ist alles richtig angeschlossen, geben Sie dieses kurze Programm ein – es schaltet die Lampe für fünf Sekunden ein und danach sofort wieder aus.

```
10 REM NETZRELAIS-TEST ACORN B
20 DDR=&FE62:DATREG=&FE60
30 ? DDR=255:REM NUR AUSGABE
40 ?DATREG=1:REM LICHT EIN
50 TIME=0:REM STARTET TIMER
60 REPEAT
70 UNTIL TIME>500
80 ?DATREG=0:REM LICHT AUS
```

```
10 REM NETZRELAIS-TEST C 64
20 DDR=56579:DATREG=56577
30 POKE DDR,255:REM NUR AUSGABE
40 POKE DATREG,1:REM LICHT AN
50 T=TI:REM STARTET TIMER
60 IF (TI-T)<300 THEN 60
70 POKE DATREG,0:REM LICHT AUS
```

Sollte die Lampe beim Programmablauf nicht leuchten, müssen Sie die Schaltung überprüfen. Ziehen Sie zuvor sämtliche Stecker!



Funktionsprinzip

Das Netzrelaismodul schützt den Computer – und auch Sie selbst – vor der hohen Netzspannung. Dazu wird der Schaltvorgang über eine ungefährliche Signalspannung ausgelöst. Computer und Netzspannung sind nur über ein Magnetfeld, nicht aber elektrisch miteinander verbunden.

Damit das Relais nicht überlastet wird, ist in der Schaltung eine 6-Ampere-Feinsicherung vorgesehen, die in einem Sicherungshalter auf einer kleinen Extraplatine im Gehäuse sitzt. Wie die Platinen genau aussehen müssen, hängt davon ab, wo beim verwendeten Relais die Signalanschlüsse beziehungsweise die (geschalteten) Netzanschlüsse liegen.

Falls Sie bisher noch nie mit Netzspannungsgeräten gearbeitet haben, sollten Sie die Schaltung nicht allein aufbauen – das könnte gefährlich werden. Wenn Sie das niedervoltgesteuerte Netzrelais nicht fertig kaufen wollen, sollten Sie Ihre Schaltung vor der ersten Inbetriebnahme noch einmal einem Fachmann – etwa einem Elektriker oder Fachhändler – zeigen. Er sieht sofort, ob alles elektrisch sicher aufgebaut ist.

Morse-Code

Morse Code

A	.–
B	–...
C	–.–.
D	–..
E	.
F	..–.
G	––.
H	....
I	..
J	.–––
K	–.–
L	.–..
M	––
N	–.
O	–––
P	.––.
Q	––.–
R	.–.
S	...
T	–
U	..–
V	...–
W	.––
X	–..–
Y	–.––
Z	––..
.	.–.–.–
,	––..––

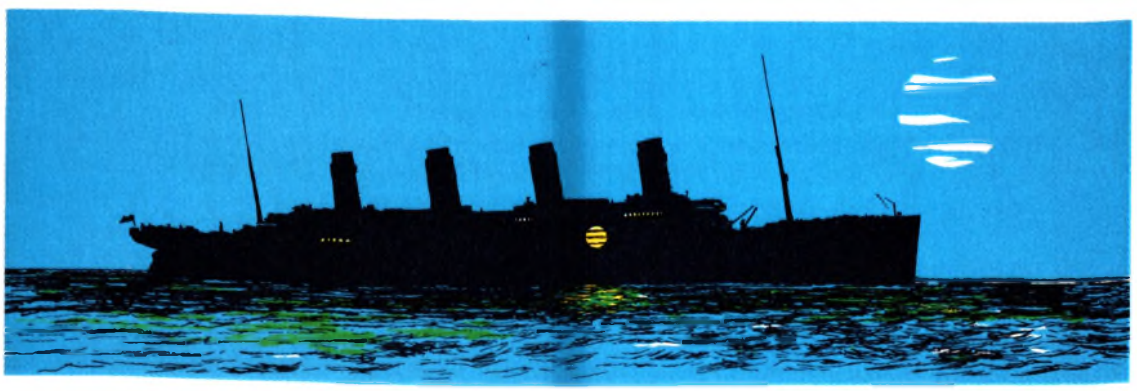
Morse-Code-Programm

Dieses Listing für den Commodore 64 läßt Ihren Rechner im Morse-Code „sprechen". Sie geben eine Nachricht (message) ein, und der Computer setzt diese in Beep-Töne und Leuchtzeichen um. Auf diese Art können Sie gleich ausprobieren, ob das Netzrelais die Signale richtig an die Lampe weiterleitet.

```
100 GOSUB 2000:          REM  INIT
150 FOR L=0 TO 1 STEP 0
200 PRINT"ENTER YOUR MESSAGE"
220 PRINT"TYPE 'BYE' TO QUIT"
240 INPUT"MESSAGE ";M$
300 ML=LEN(M$):M$=""
320 FOR K=1 TO ML
330 C$=MID$(M$,K,1)
340 IF C$=)"A"ANDC$(="Z"THEN M$=M$+C$
350 IF C$=" " THEN M$=M$+C$
360 NEXT K:IF M$="" THEN NEXT L
400 ML=LEN(M$)
420 FOR K=1 TO ML
440 CH$=MID$(M$,K,1):CH=ASC(CH$)-64
450 IF CH=-32 THEN FORPP=1TO6*DE:NEXTPP
460 IF CH()-32 THEN GOSUB 3000
480 FOR PP=1 TO 3*DE:NEXT PP
500 NEXT K
550 IF M$="BYE" THEN L=1
600 NEXT L
900 END
2000 REM*******INIT******************
2100 DIM M$(26)
2110 DDR=56579:DATRG=56577:POKE DDR,255
2120 DE=25:RX=2*DE
2130 V=54296:LF=54272:HF=54273:W=54276
2140 A=54277:S=54278
2150 FOR K=LF TO LF+24:POKEK,0:NEXT K
2160 POKEA,24:POKES,129:POKEV,15
2200 DATA ".-","-...","-.-.","-..","."
2220 DATA "..-.","--.","....",".."
2240 DATA ".---","-.-","-.-.","--"
2260 DATA "-.","---",".--.","--.-"
2280 DATA ".-.","...","-","..-","...-"
2300 DATA ".--","-..-","-.--","--.."
2400 FOR K=1 TO 26:READ M$(K):NEXT K
2990 RETURN
3000 REM*********FLASH & BEEP********
3050 PRINT CH$,M$(CH)
3100 N=LEN(M$(CH))
3200 FOR C=1 TO N
3220 D=DE-(ASC(MID$(M$(CH),C,1))-46)*RX
3240 POKE DATRG,1          :REM FLASH
3250 POKE LF,172:POKE HF,57:REM BEEP
3260 POKE W,33:FOR PP=1 TO D:NEXT PP
3270 POKE W,32
3280 POKE LF,0:POKE HF,0    :REM UNBEEP
3290 POKE DATRG,0           :REM UNFLASH
3300 FOR PP=1 TO DE:NEXT PP
3320 NEXT C
3490 RETURN
```

Auch dieses Listing dient dazu, geschriebene Wörter in den Morse-Code zu übersetzen. Es ist für den Acorn B. Jeder beliebige Alpha-String wird mit Hilfe von Licht- und Tonsignalen wiedergegeben. Wenn Sie auch Zahlen umsetzen wollen, müßten die entsprechenden Morse-Codes noch als Datas eingegeben werden.

```
 100 PROCinitialise
 120 ALLOVER=FALSE
 150 REPEAT
 200 PRINT"ENTER YOUR MESSAGE"
 220 PRINT"TYPE 'BYE' TO QUIT"
 240 INPUT"MESSAGE",MS$
 300 ML=LEN(MS$):M$=""
 320 FOR K=1 TO ML
 330 C$=MID$(MS$,K,1)
 340 IF C$)="A"ANDC$(="Z"THEN M$=M$+C$
 350 IF C$=" " THEN M$=M$+C$
 360 NEXT K:IF M$="" THEN UNTIL FALSE
 400 ML=LEN(M$)
 420 FOR K=1 TO ML
 440 CH$=MID$(M$,K,1):CH=ASC(CH$)-64
 450 IF CH=-32 THEN PROCdelay(6*DE*IX)
 460 IF CH()-32 THEN PROCbeepflash
 480 PROCdelay(3*DE)
 500 NEXT K
 550 IF M$="BYE" THEN ALLOVER=TRUE
 600 UNTIL ALLOVER
 900 END
2000 REM**********INIT*************
2050 DEFPROCinitialise
2100 DIM M$(26)
2110 DDR=&FE62:DATREG=&FE60:?DDR=255
2120 DE=3:RX=2*DE:IX=30
2200 DATA ".-","-...","-.-.","-..","."
2220 DATA "..-.","--.","....",".."
2240 DATA ".---","-.-","-.-.","--"
2260 DATA "-.","---",".--.","--.-"
2280 DATA ".-.","...","-","..-","...-"
2300 DATA ".--","-..-","-.--","--.."
2400 FOR K=1 TO 26:READ M$(K):NEXT K
2990 ENDPROC
3000 REM*****FLASHANDBEEP**********
3020 DEFPROCbeepflash
3050 PRINT CH$,M$(CH)
3100 N=LEN(M$(CH))
3200 FOR C=1 TO N
3220 D=DE-(ASC(MID$(M$(CH),C,1))-46)*RX
3240 ?DATREG=1              :REM  FLASH
3260 SOUND 1,-15,200,D      :REM  BEEP
3270 PROCdelay(IX*D)
3290 ?DATREG=0              :REM  UNFLASH
3300 PROCdelay(DE*IX)
3320 NEXT C
3490 ENDPROC
4000 REM*******DELAY***************
4100 DEFPROCdelay(time)
4200 FOR DD=1 TO time:NEXT DD
4490 ENDPROC
```



Kritischer Pfad

Die Planung eines vielschichtigen Projekts wird weitaus übersichtlicher, wenn die Aufgaben in Diagrammform dargestellt werden. Das Programm MacProject für den Macintosh bietet auch dem Anfänger die Möglichkeit, geordnete und übersichtliche Projektpläne zu entwerfen.

Die „Analyse des kritischen Pfades" (Critical Path Analysis – CPA) ist ein vielseitiges Managementinstrument, mit dem sich Projekte in einzelne Schritte unterteilen lassen. Jeder Schritt wird durch einen Pfeil dargestellt, der auch den Zeitbedarf anzeigt. So entsteht eine Reihe paralleler Ketten, die an bestimmten Punkten zusammentreffen. Die Kette mit dem größten Zeitbedarf ist der „kritische Pfad" – er bestimmt die Gesamtdauer des Projekts. Aus den Diagrammen läßt sich ablesen, wann und wo welche Materialien, Werkzeuge etc. benötigt werden, wann einzelne Schritte begonnen und beendet werden müssen und wer sie durchführt.

Es wurden schon mehrere Versuche unternommen, CPA-Programme zu erstellen. Da Netzwerkdiagramme jedoch hauptsächlich aus grafischen Elementen bestehen, gab es beim Einfügen von Informationen Probleme mit der Neuberechnung der Grafikpositionen. CPA-Programme waren daher umständlich, unübersichtlich und konnten oftmals nur Text darstellen. Damit wurde jedoch genau das Gegenteil erreicht: Netzwerkdiagramme sollen Abläufe durchsichtiger machen, nicht komplizierter.

Das Grafiksystem des Macintosh eignet sich ideal für diese Aufgabe. Mit MacProject ist der Aufbau eines Netzwerks (im Programm „Project Chart" genannt) so leicht, daß sich die Steuerung eines Projekts erheblich vereinfacht. Die Daten und Zeiten lassen sich beliebig ändern, wobei die jeweiligen Projektzeiten automatisch neu berechnet werden. Neue Aufgaben können eingefügt und nicht mehr aktuelle Abläufe gelöscht werden.

Das knappe, leicht verständliche Handbuch erklärt das am Beispiel einer Grillparty:

SALAT VORBEREITEN
TISCH DECKEN
FEUER ANZÜNDEN
REIS KOCHEN
STEAKS GRILLEN

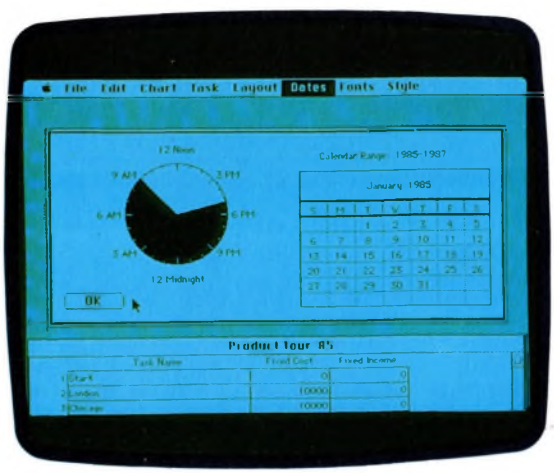
Das Ergebnis des Projekts ist:
MAHLZEIT EINNEHMEN

Die meisten Aufgaben lassen sich gleichzeitig erledigen. Da die Steaks jedoch nicht vor dem Anzünden des Feuers gegrillt werden können, führt der kritische Pfad von „Feuer anzünden" über „Steaks grillen" zu „Mahlzeit einnehmen". Will das Feuer nicht brennen, so verzögert sich das Essen, selbst wenn die anderen Aufgaben erledigt sind. Auch die Salatvorbereitung kann zum kritischen Pfad werden, wenn sich der Helfer verspätet.

Probleme ähneln einander

Nun braucht man zwar für die Planung einer Grillparty nicht unbedingt ein Programm für circa 500 Mark, aber ähnliche Elemente lassen sich auch für jedes andere Projekt einsetzen: von der Organisation einer Konferenz bis zum Aufbau einer Öl-Raffinerie.

Mit der Maus werden Aufgabenkästen gezeichnet, wobei die Zeitrechnung automatisch am 2. Januar beginnt. Dieses Anfangsdatum läßt sich jedoch ändern oder vom Endpunkt aus rückwärts berechnen. Die Größe des ersten Kastens gilt automatisch für alle weiteren Kästen. Um Verbindungslinien zu zeichnen, setzt man die Maus zuerst in das Innere des Kastens und dann auf die gewünschte nächste Position. In die Kästen lassen sich Aufgabenbezeichnungen eintragen, die automatisch zentriert werden. Die Kästen lassen sich vergrößern, indem die Maus auf die Umrißlinie gesetzt und die Taste gedrückt wird. Dabei zeigt das Programm acht verschiedene Größen, die wiederum mit der Maus angewählt werden können. Wenn der Cursor auf die Umrißlinie gesetzt, aber keine neue Größe angewählt wird, läßt sich der Kasten an jede Stelle des Diagramms stellen.



Über den Kalender des MacProject – hier gezeigt als oberes Bildschirmfenster – lassen sich die Anzahl der Arbeitsstunden und Tage eintragen, die für die Fertigstellung des Projekts benötigt werden. Dabei werden außerdem Urlaubsperioden, Wochenenden und Festtage berücksichtigt. Am unteren Bildschirmrand ist ein Teil des Kassenbuchs zu sehen, in dem die laufenden Ausgaben des Projekts eingetragen werden.

Die Wahl der Option INVISIBLE GRID (unsichtbares Raster) – Menüpunkt LAYOUT – richtet alle Kästen exakt aus.

Nachdem ein Kasten mit Namen versehen wurde, kann er mit Informationen (TASK INFO) gefüllt werden. Dabei lassen sich die Zeit (Minuten, Tage, Wochen etc.) und die Hilfsmittel (beispielsweise Materialien) angeben. Das Hauptdiagramm zeigt für jeden Kasten vier Informationen: links oben Hilfsmittel oder frühester Anfangstermin; rechts oben Dauer, Hilfsmittel oder frühester Endtermin; unten links letzter Anfangstermin, feste Kosten oder Hilfsmittel; unten rechts letzter Endtermin, feste Einnahmen oder Hilfsmittel. Der kritische Pfad erscheint dabei mit starker Umrahmung, nicht-kritische Wege dagegen mit dünnen Linien.

Parallel zum Hauptdiagramm entsteht eine Reihe von Analysediagrammen. Die Option RESOURCE TIMELINE zeigt in einem Balkendiagramm die Verteilung der Hilfsmittel. TASK TIMELINE enthält die gleiche Information, jedoch nach Aufgabenbereichen geordnet. „Slack Time" (die Differenz zwischen dem frühesten und dem letztmöglichen Endtermin – also die Zeit, in der die Aufgabe abgeschlossen werden muß, wenn das Projekt nicht verzögert werden soll) wird in Grau angezeigt. Die Projekttabelle stellt eine Liste aller Aufgaben dar (die kritischen Bereiche in Fettdruck). Sie enthält außerdem für jede Aufgabe die Anzahl der zugeteilten Tage, den frühesten und letztmöglichen Anfangs- und Endtermin, feste Kosten, feste Einnahmen und die Kosten und Bezeichnungen der Hilfsmittel. Drei Tabellen geben übersichtlich und auf einen Blick Auskunft über die Kosten: TASK COST ENTRY (Projektkosten), RESOURCE COST ENTRY (Hilfsmittelkosten) und CASH FLOW TABLE (Kassenbuch).

Über die Menüoption LAYOUT läßt sich bestimmen, ob ein schematisches Diagramm des gesamten Netzwerks (ohne Erläuterungen) dargestellt oder die Größe der Darstellung verändert werden soll. Mit dieser Option läßt sich die Netzwerkdarstellung komprimieren, wenn das Diagramm nach dem Löschen einzelner Kästen noch leere Plätze anzeigt.

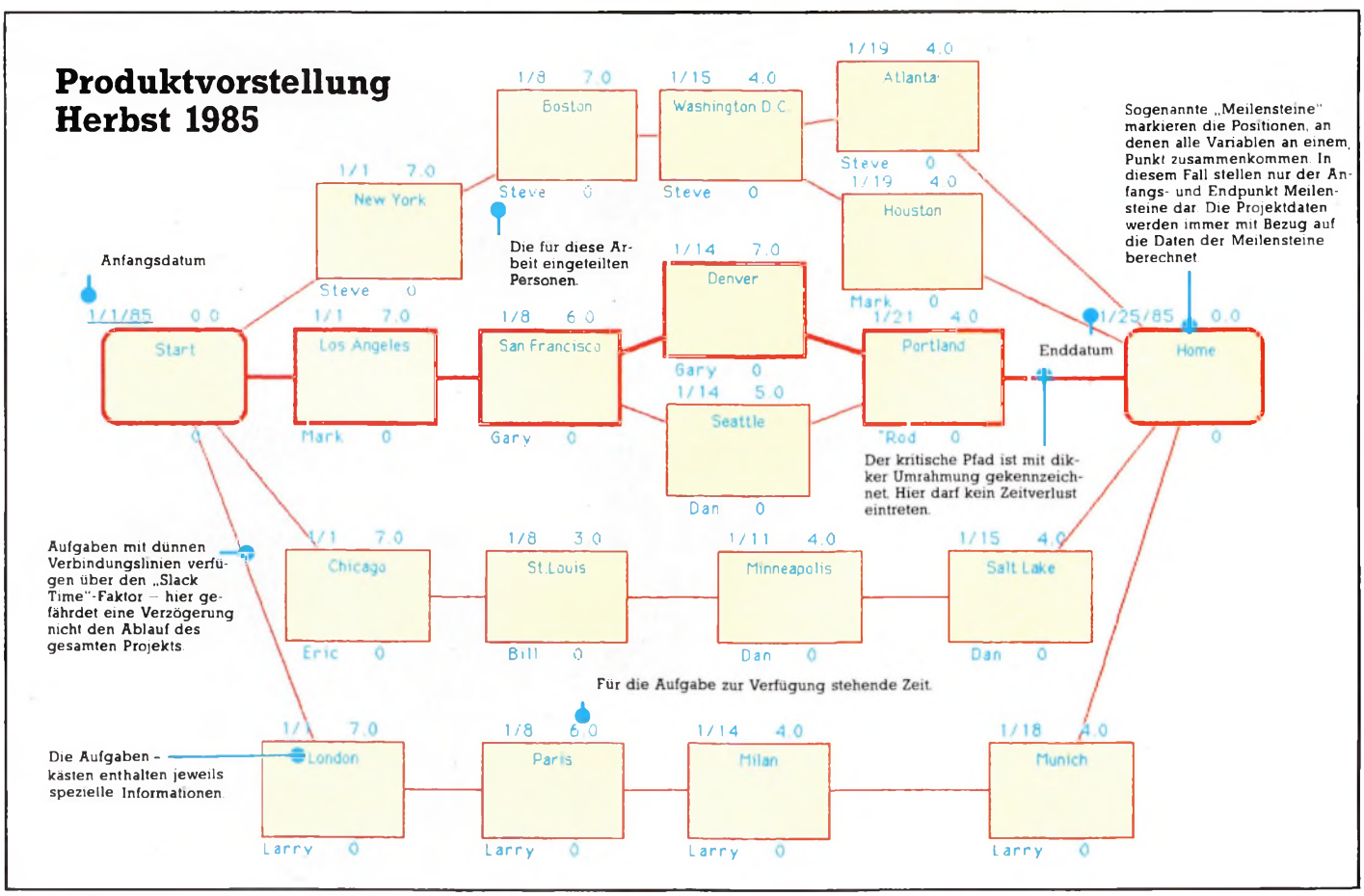
In MacProject ist die Analyse des kritischen Pfades sehr gut grafisch umgesetzt worden, so daß dies auch der Anfänger leicht verstehen kann. Die Grafikfähigkeiten des Macintosh wurden dabei so umfassend eingesetzt, daß ein ähnliches Programm auf anderen Geräten kaum vorstellbar ist. MacProject wird ebenfalls in einer deutschen Version angeboten.

MacProject: Für den Apple Macintosh

Kontaktadresse: Apple Computer GmbH, Ingolstädter Straße 20, 8000 München 45

Autoren: Debra Willet und Stephen D. Young

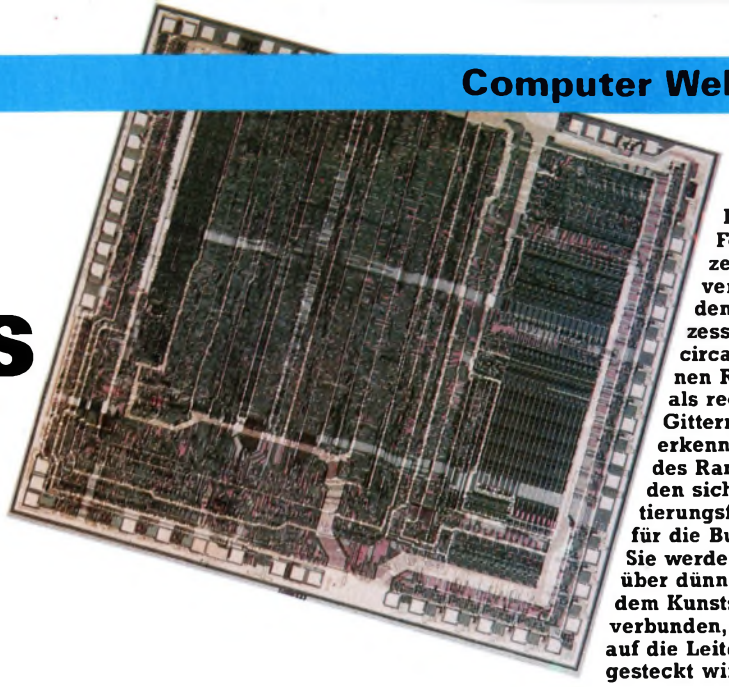
Format: Diskette



Dieses Diagramm von MacProject stellt eine Tour durch mehrere Städte dar, auf der eine Firma ein neues Produkt vorstellt. Durch die visuelle Darstellung gestaltet sich die Zeitplanung und Aufteilung der Fachkräfte außerordentlich einfach.

Chip-Tourismus

Das komplizierte Zusammenwirken der verschiedenen Komponenten eines Microcomputers ist nicht ganz leicht zu verstehen. Eine Platinen-„Rundreise" soll die Funktion der einzelnen Bauteile und ihr Wechselspiel erklären.



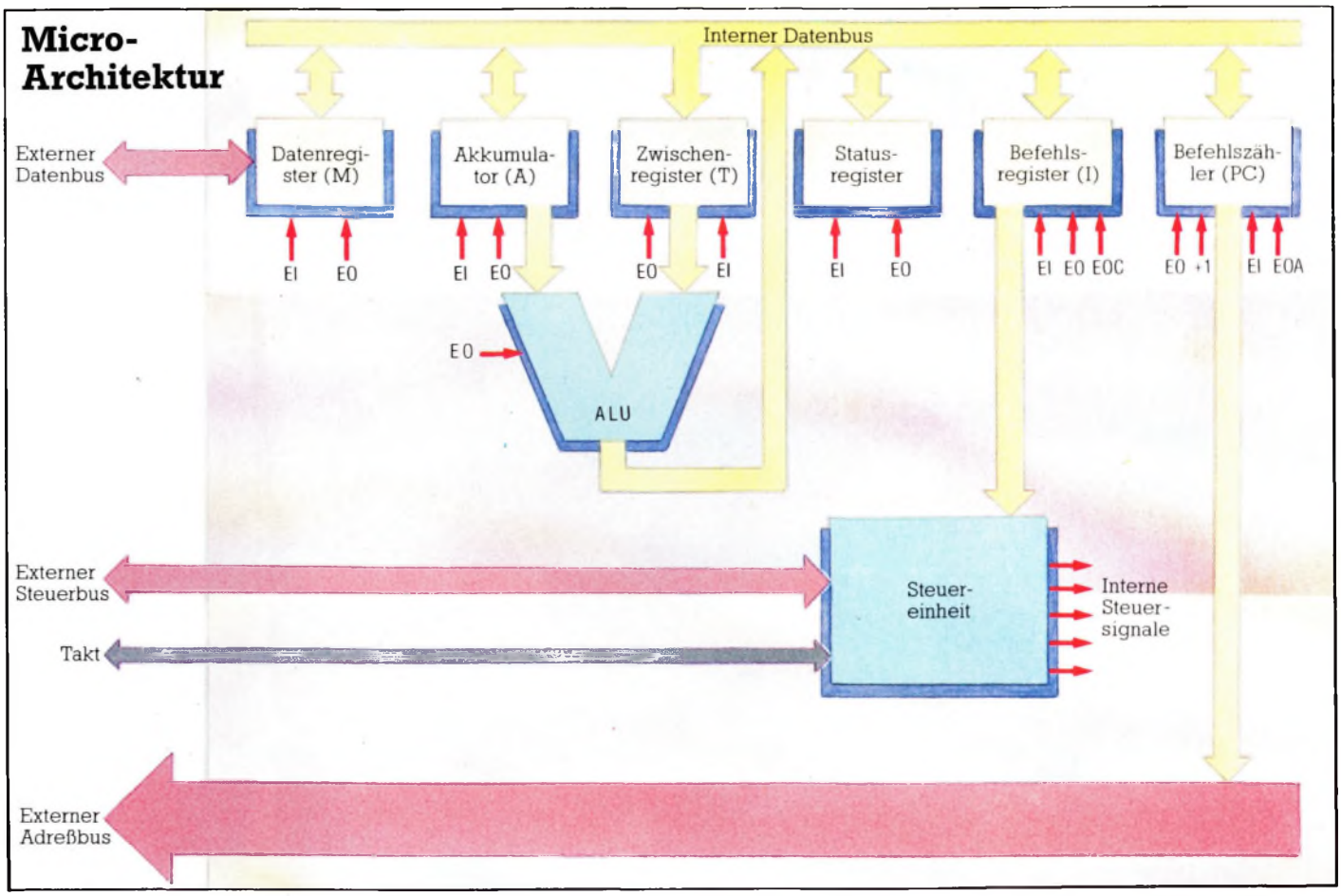
Diese Fotografie zeigt stark vergrößert den Zilog-Prozessor Z80. Die circa 25 internen Register sind als rechtwinklige Gittermuster zu erkennen. Längs des Randes befinden sich 40 Kontaktierungsflächen für die Busanschlüsse. Sie werden später über dünne Drähte mit dem Kunststoffgehäuse verbunden, das dann auf die Leiterplatte gesteckt wird.

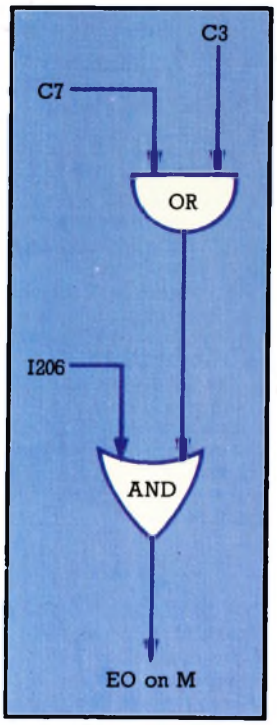
Unter dem Mikroskop sind auf einem Microprozessorchip typische Strukturen zu erkennen: Die internen Register erscheinen als charakteristische Gitterfelder, die Logikgatter als unregelmäßig gemusterte Bereiche, und die quadratischen Metallplättchen am Rande sind für die Kontaktierung, also die Verbindung mit der Außenwelt zuständig. Außerdem fallen Gruppen paralleler Leitungen auf, die der internen Kommunikation zwischen den Funktionselementen dienen.

Jede CPU-Anweisung setzt eine ganze Kette von elektronischen Elementarschritten in Gang, die jenseits der Software-Zuständigkeit ablaufen. Zur Illustration wollen wir beschreiben, wie auf einen ADD-Befehl hin die Addition einer Zahl zum Akkumulatorinhalt mit Rückschreiben der Summe abläuft.

Das untenstehende Diagramm zeigt in vereinfachter Form die typische CPU-Architektur. Alle Register kommunizieren über einen internen Datenbus und hängen außerdem zur Synchronisation des Datenverkehrs über Steuerleitungen an einer zentralen Steuerlogik.

Die Abbildung veranschaulicht den Aufbau eines typischen 8-Bit-Prozessors. Die Funktionsgruppen kommunizieren miteinander über interne Datenbus- und Steuerleitungen. Über Steuersignale und den externen Daten- und Adreßbus hat die CPU Verbindung mit dem Arbeitsspeicher und der Peripherie.





Für den ADD-Befehl muß in den Schritten 3 und 7 des Befehlszyklus das Steuersignal „Enable Output" (EO) an das Datenregister (M) gesendet werden. Diese Gatter verknüpfen die Leitungen C2 und C3 des Takt-Decoders so mit dem Befehlsdekoder-Ausgang I206, daß zum richtigen Zeitpunkt das EO-Signal am M-Register ansteht.

Der erwähnte ADD-Befehl steht vorerst in mehreren aufeinanderfolgenden Bytes innerhalb eines längeren Programms im Speicher des Rechners. Das erste Byte enthält dabei den Operationsteil, das zweite die zu addierende Zahl. Der Prozessor ruft nun zunächst das Byte mit dem Opcode aus dem Arbeitsspeicher ins Befehlsregister ab („Fetch"-Zyklus) und führt dann die Anweisung aus („Execute"-Zyklus). Die Adresse des Befehlsbytes steht schon vorher im Befehlszähler (Programm Counter = PC) bereit. Beim Fetch-Execute-Zyklus ist von besonderem Interesse, wie die Steuereinheit den Datenaustausch zwischen den Registern organisiert – das jeweilige Register muß im richtigen Moment ein Freigabe(Enable)-Signal bekommen. Für diese Signale sind folgende Abkürzungen zweckmäßig:

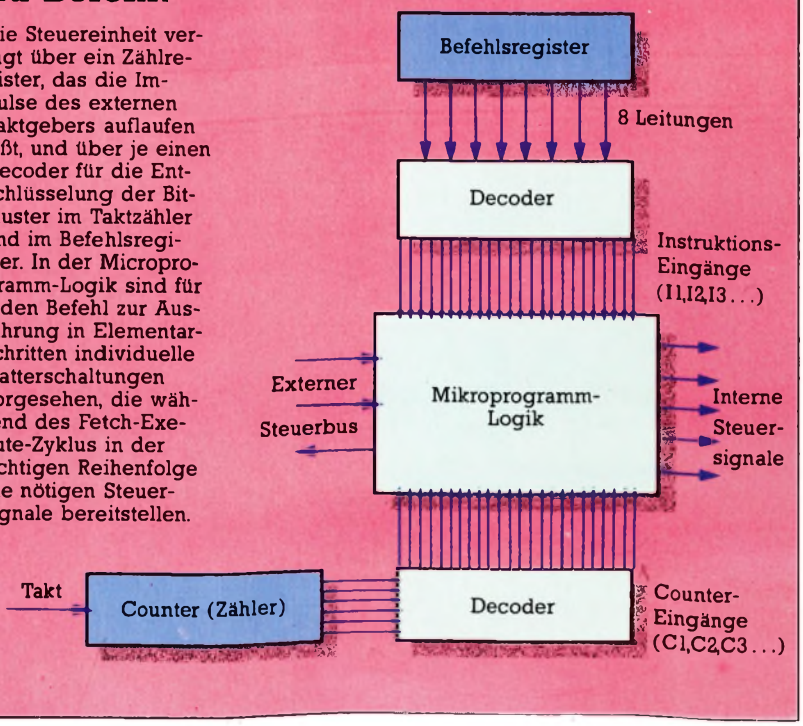
EI (= Enable Input) gibt ein Register für das Beschreiben vom internen Datenbus aus frei,
EO (= Enable Output) veranlaßt das Auslesen eines Registers auf den Datenbus,
EOC (= EO to Control Unit) bewirkt die Ausgabe des Registerinhalts an die Steuerlogik,
EOA (= EO to Address Bus) das Auslesen des Registers auf den Adreßbus.

Der Fetch-Zyklus umfaßt vier Schritte:

Schritt	Steuersignal	Wirkung
1	EOA an PC	Kopieren des Befehlszählers(PC) auf den Adreßbus
2	+1 an PC	Erhöhen des Befehlszählers
3	EO an M	Kopieren des Datenregisters (M)
	EI an I	ins Befehlsregister(I)
4	EOC an I	Kopieren des Befehlsregisters in die Steuerlogik

Zu Befehl!

Die Steuereinheit verfügt über ein Zählregister, das die Impulse des externen Taktgebers auflaufen läßt, und über je einen Decoder für die Entschlüsselung der Bitmuster im Taktzähler und im Befehlsregister. In der Microprogramm-Logik sind für jeden Befehl zur Ausführung in Elementarschritten individuelle Gatterschaltungen vorgesehen, die während des Fetch-Execute-Zyklus in der richtigen Reihenfolge die nötigen Steuersignale bereitstellen.



Mit dem vierten Schritt ist endlich der ADD-Befehl in die Steuerlogik gebracht, und der Befehlszähler enthält die Adresse des Bytes mit der Konstanten, die addiert werden soll. Vor der Erklärung des Execute-Zyklus ist noch ein Blick auf die Steuereinheit angebracht.

Geordnetes Hin und Her

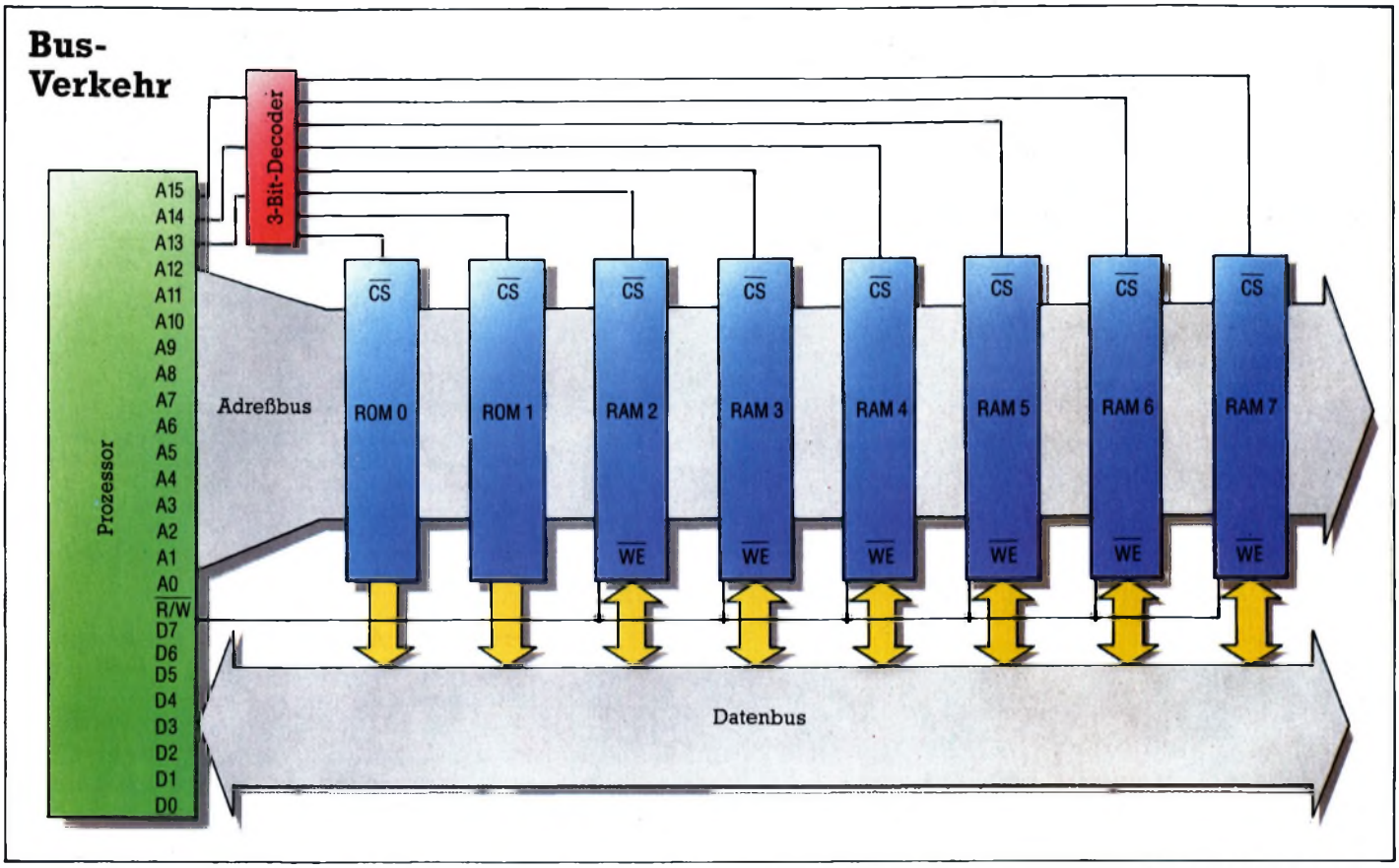
Sie enthält ein Register für die Aufnahme des aktuellen Opcodes und einen Decoder, der entsprechend dem 8-Bit-Muster des Befehls eine von 256 Ausgangsleitungen aktiviert – wenn nicht alle möglichen Opcodes genutzt werden, sind es weniger als 256 Leitungen. Darüber werden nun in der „Microprogrammlogik" individuelle Gatterschaltungen angewählt, durch die im Zusammenspiel mit einem Taktzähler-Register (Counter) und einem weiteren Decorder die verschiedenen Steuersignale bereitgestellt werden. Dabei läuft hardwaremäßig ein „Microprogramm" ab; in unserem Beispiel liefern die Taktschritte während der fünf Taktschritte des Execute-Zyklus folgende Signale:

Schritt	Steuersignal	Wirkung
5	EOA an PC	Einlesen der Additions-
	EI an M	konstanten in M
6	+1 an PC	PC für das nächste Befehls-
		byte inkrementieren
7	EO an M	Konstanteneingabe in das
	EI an T	ALU-Zwischenregister (T)
8	ADD an ALU	Addieren der Konstanten zum
	EO an T u. A	Akku-Inhalt in der ALU
9	EO an ALU	Ergebnisübernahme in den
	EI an A	Akkumulator

Während der Fetch-Zyklus für alle Befehle gleich bleibt, muß der Execute-Zyklus genau auf die jeweilige Anweisung zugeschnitten sein – hier auf das Laden der Konstanten und ihre Addition zum Akku-Inhalt. Deshalb erfordert jeder Opcode ein individuelles Microprogramm, das heißt eine eigene Gatterschaltung. Wie damit die Steuersignale erzeugt werden, läßt sich am besten an einem konkreten Beispiel erklären: Das Ausgabesignal für das Datenregister (EO an M) muß bei Schritt 3 und 7 des Befehlszyklus aktiviert werden. Der Opcode für die Konstantenaddition ist 1100, 1110, dezimal also 206. Sie brauchen nun nur die Leitungen C3 und C7 vom Taktregister-Dekoder durch ein OR-Gatter zu verknüpfen und das Ergebnis mit der Leitung I206 vom Befehlsdekoder zu ANDen – schon erhält man die gewünschten EO-Signale. Das M-Register muß natürlich auch noch durch andere Microprogramme ansprechbar sein, aber das läßt sich einfach durch OR-Verknüpfungen sämtlicher EO-Signale erreichen.

Näheres über diese „Microprogrammierung" werden wir Ihnen berichten.

Schreiben und Lesen



Nachdem im vorigen Heft erklärt wurde, wie der Befehlszyklus innerhalb der CPU durch die Steuerlogik abgewickelt wird, geht es jetzt um die Speicherbausteine, und zwar vor allem um die Schreib- und Lesevorgänge.

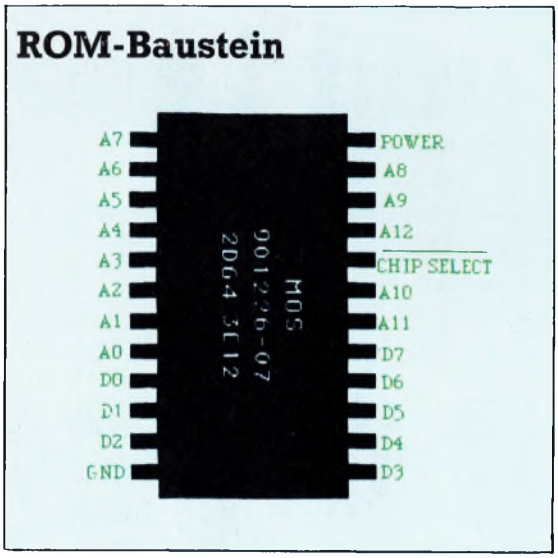
Das Blockdiagramm zeigt eine typische Speicherstruktur für einen 8-Bit-Prozessor mit 16 K ROM und 48 K RAM (statisch). Der Schreib/Lese-Ausgang R/W des Prozessors ist mit den Schreibfreigabe (WE)-Anschlüssen sämtlicher RAM-Chips verbunden. Wenn die WE-Leitung durch die CPU auf Null gesetzt wird, erfolgt die Datenübernahme in das RAM-Byte, das gerade adressiert ist. Zur Auswahl eines bestimmten RAM- oder ROM-Bausteins werden die drei höchsten Adreßbits durch einen Decoder auf eine von acht Chip-Select-Leitungen umgesetzt.

Den Unterschied zwischen einem RAM und einem ROM kennt fast jeder, der überhaupt mal etwas von Hardware gehört hat. Während ein RAM nicht nur ausgelesen, sondern auch beschrieben werden kann, ist der ROM-Inhalt durch den Herstellungsprozeß fixiert. Da bei den RAMs zwischen „statischen" und „dynamischen" zu unterscheiden ist, vorweg noch eine kurze Bemerkung dazu: Bei den statischen RAMs sind die eigentlichen Speicherzellen J-K-Flipflops, die jeweils für ein Bit zuständig sind. Ein Flipflop hält einen definierten Ausgangswert (1 oder 0) fest, bis er durch einen neuen Eingangsimpuls „umgekippt" wird. Für jedes Speicherbyte sind acht derartige Flipflops vonnöten – ein statischer RAM-Baustein enthält also viele Tausende davon.

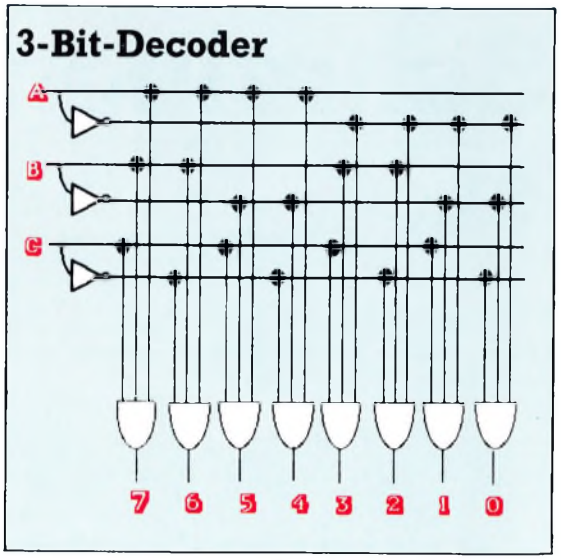
Bei einem dynamischen RAM dagegen sind die Datenbits in Form elektrischer Ladungen gespeichert. Das erfordert zwar viel weniger Platz, so daß der einzelne Chip eine größere Kapazität aufweist, aber dafür muß ein dynamisches RAM periodisch „aufgefrischt" werden, weil die Ladung infolge von Leckströmen sehr schnell abfließt.

Die nächste Seite zeigt die Pinbelegung des Prozessorbausteins 6510, der vom 6502 abstammt, und daneben ein typisches 8-KByte-ROM. Der Prozessor hat vier Gruppen von Anschlüssen, nämlich einen 16-Bit-Adreßbus (A0-A15), einen 8-Bit-Datenbus (D0-D7) und eine Anzahl von Steuerleitungen. Als Besonderheit kommt beim 6510 noch ein 6-Bit-Port hinzu, über den unter anderem zusätzliche RAM- oder ROM-Bänke in den 64K-Adreßraum geschaltet werden können.

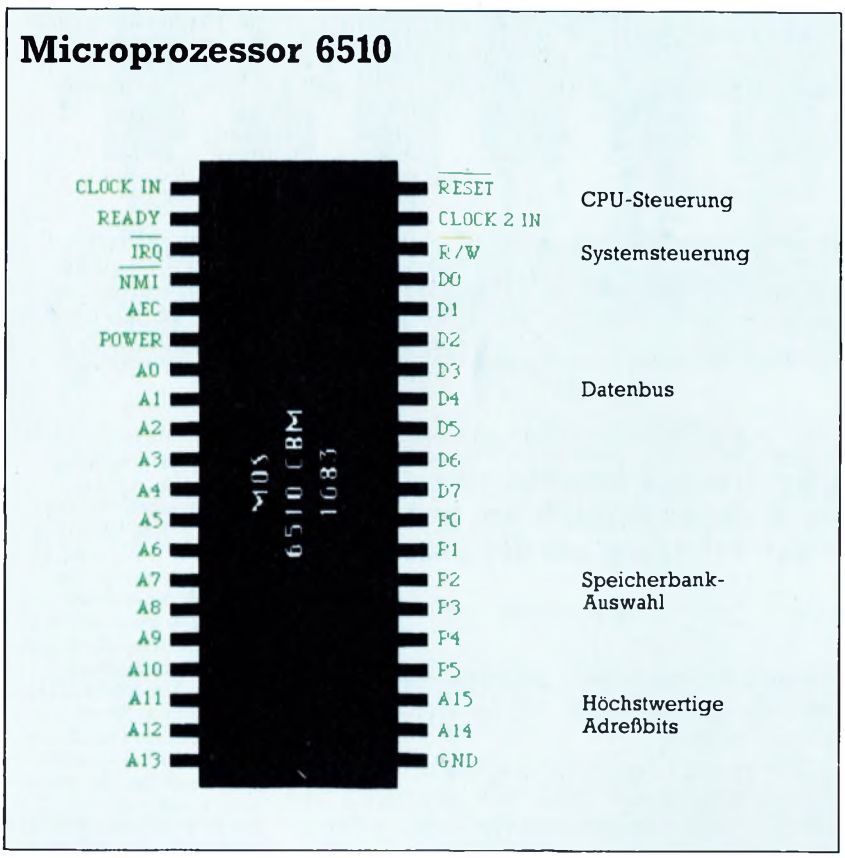
Das daneben abgebildete 8K-ROM hat 13 Adreßleitungen (A0-A12) für das Anwählen der 8192 Bytes, acht Datenleitungen und einen Chip-Select-Anschluß. Statische RAMs weisen eine analoge Pinbelegung auf, haben aber zusätzlich noch einen „Write Enable"-Pin. Der Schlüssel zur Adressierung solcher RAMs und ROMs ist die Chip-Select-Leitung.



Das hier gezeigte 24-polige 8K-ROM weist eine typische Pinbelegung auf: 13 Eingänge werden zu Adressierung der 8192 Byte benötigt, acht Pins beansprucht der Datenbus, zwei die Spannungsversorgung (Power und Ground=GND), und der Chip-Select-Anschluß dient zur Anwahl dieses speziellen ROMs.



Ein Decoder ist eine Digitalschaltung zur Entschlüsselung von Binärzahlen. Jedem Bitmuster am Eingang ist genau eine Ausgangsleitung zugeordnet, der gezeigte 3-Bit-Decoder muß acht Ausgänge haben. Ausgang 0 wird durch den Eingangswert ABC=000 aktiviert, Ausgang 1 durch 001 usw. bis zum Ausgang 7 für ABC=111.



Wie der abgebildete 6510 haben die meisten 8-Bit-Prozessoren 40 „Beinchen", nämlich 16 für den Adreßbus, acht für den Datenbus, den Rest für Takt- und Steuersignale (einschließlich Interrupt-Eingänge und Schreibfreigabe) sowie zur Spannungsversorgung. Der 6510 ist eine verbesserte 6502-Version. Der Fortschritt besteht wesentlich in einem zusätzlichen Datenregister für die Cassetten-Ein/Ausgabe.

Ein 8-Bit-Prozessor kann über den 16-Bit-Adreßbus bis zu 64 KByte direkt ansprechen (2^{16}=65536=64 K). Es sind also acht separate RAM- oder ROM-Chips mit je acht KByte adressierbar. Dabei werden die Adreßleitungen A0-A12 für die Ansteuerung des einzelnen Bytes innerhalb eines Bausteins beansprucht, während die Leitungen A13-A15 für die Chip-Auswahl zur Verfügung stehen. Als Beispiel ist ein typisches Speicherkonzept mit zwei 8K-ROMs und sechs 8K-RAMs abgebildet. Die drei obersten Bits des Adreßbus werden für die Chip-Auswahl auf einen 3-Bit-Decoder mit acht Ausgängen gegeben, während die unteren 13 Adreßbits bei allen Chips gleichzeitig anstehen. In Aktion tritt aber immer nur der durch das Chip-Select-Signal jeweils angewählte Baustein.

Bei den RAMs wird über die ROM-Anschlüsse hinaus noch ein „Read/Write"-Signal zur Unterscheidung von Lese- und Schreibvorgang benötigt. Deshalb hat jedes RAM noch einen Write-Enable-Pin, der über Low-Pegel aktiviert wird und deshalb die Bezeichnung $\overline{WE}$ trägt; er ist mit dem Read/Write-Kontakt R/$\overline{W}$ der CPU zu verbinden. Beim 6510 liegt die R/$\overline{W}$-Leitung normalerweise auf High und wird nur zum Schreiben auf Low heruntergesetzt.

Gewußt, wo

Wenn der Prozessor etwa den Lesebefehl ausführen soll „Akkumulator mit dem Inhalt von Speicherplatz $C000 laden", legt er zunächst die hexadezimale Adresse $C000 auf den Adreßbus, während die R/$\overline{W}$-Leitung auf High bleibt. Die drei höchsten Adreßbits haben in diesem Fall den Wert 110, das heißt, RAM 6 in unserem Blockdiagramm wird durch den 3-Bit-Decoder angewählt. Die übrigen 13 Adreßbits sind alle Null, so daß das unterste Speicherbyte von RAM 6 angesprochen ist. Die Information gelangt über den Datenbus in CPU und Akkumulator.

Für eine Write-Anweisung, etwa „Akku-Inhalt unter $C000 abspeichern", erfolgt genau der gleiche Adressiervorgang, aber außerdem wird die R/$\overline{W}$-Leitung und damit der Schreibfreigabe-Anschluß $\overline{WE}$ heruntergezogen. Der Akku-Inhalt steht bereits auf dem Datenbus an und wird mit dem Freigabesignal in das erste Byte von RAM 6 kopiert.

Dynamische RAMs

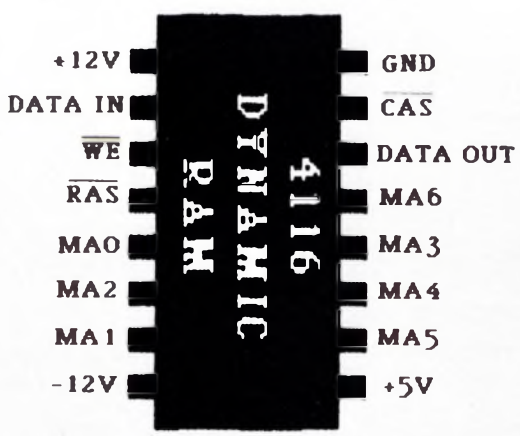
In der vorigen Folge wurde erklärt, wie ROM und statische RAMs vom Prozessor gesteuert werden. Heute untersuchen wir dynamische RAMs und gewinnen Einblick in die Arbeitsweise eines RAM-Chips.

In der letzten Folge hatten wir kurz erwähnt, daß es zwei Arten von RAMs gibt: dynamische und statische. Während statische RAMs mit kleinen logischen Mechanismen – den Flipflops – arbeiten, speichern dynamische RAMs ihre Datenbits als elektrische Ladung.

Beide RAM-Typen haben Vor- und Nachteile. So ist der Transistor, der in dynamischen RAMs eine Bit-Ladung speichert, weitaus kleiner als ein Flipflop, der die gleiche Information in statischen RAMs festhält. Aus diesem Grund lassen sich dynamische RAMs dichter packen. Da die Ladung dynamischer RAMs sich jedoch schon nach einigen Millisekunden „verflüchtigt", sind Zusatzschaltungen nötig, die sie periodisch „auffrischen". Statische RAMs haben dieses Problem nicht und sind eine Lösung für Systeme mit kleinem Arbeitsspeicher.

Statische RAMs werden normalerweise in Acht-Bit-Registern angeordnet. Dabei verfügt jeder Chip über acht Datenkontakte, die mit den acht Leitungen des Datenbusses verbunden sind. Dynamische RAMs haben jedoch oft einen anderen Aufbau. Jeder RAM-Chip stellt dabei eins der acht Datenbits einer Speicherstelle dar. Der Speicher enthält dann acht parallel geschaltete Chips dieser Art.

Das Bild unten zeigt den 4116 mit den typischen Kontakten eines 16-KBit-Chips. Für einen Arbeitsspeicher von 64 KByte verwendet man bei vielen Acht-Bit-Micros den dynamischen RAM Chip 4164 (64 KBit). Obwohl wir uns den Chip als ein Bit „breit" und 16 mal 1024 Bits „lang" vorstellen können, enthält er 128 Zeilen mal 128 Spalten.



Die typische Pinbelegung eines 16-KBit RAM: Anders als beim statischen RAM korrespondiert ein dynamisches RAM mit einem Bit des Datenworts – daher je eine einfache Data-in- und Data-out-Leitung. Die Verkabelung ist recht einfach.

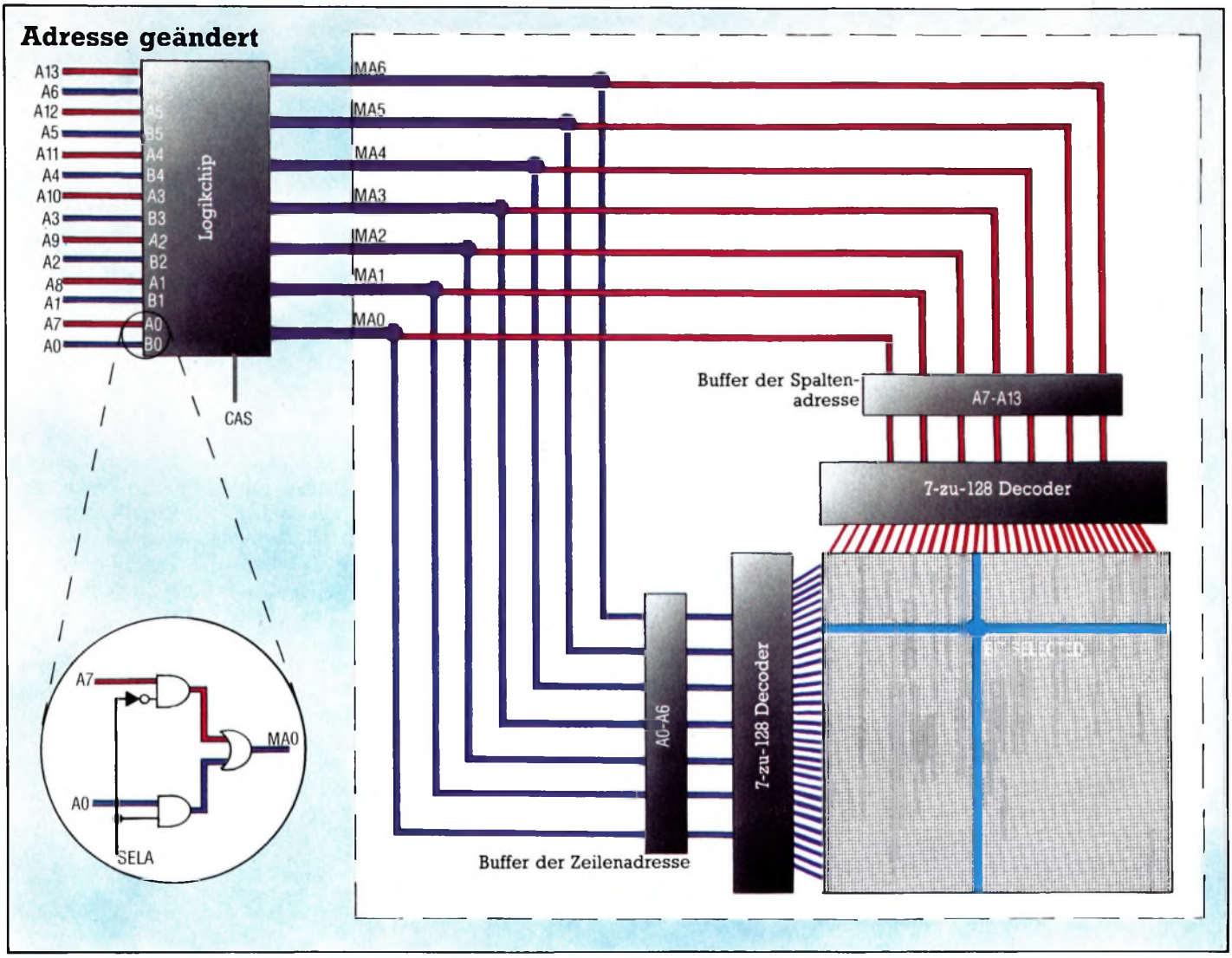
Bei einem Vergleich zwischen diesem Chip und einem statischen ROM fallen etliche Unterschiede ins Auge. Zunächst gibt es statt acht Datenkontakten nur zwei: „Data in" und „Data out". Bei einem 16-KBit-Chip wären 14 Adreßbits zu erwarten, die jedes Bit einzeln ansprechen. Der 4116 hat jedoch nur sieben. Außerdem gibt es zwei Leitungen mit seltsamen Bezeichnungen: $\overline{RAS}$ und $\overline{CAS}$ – „Row Adress Strobe" und „Column Adress Strobe". Mit diesen zwei Zeitsignalen läßt sich die Adresse in zwei Hälften darstellen (daher sieben statt vierzehn Adreßkontakte). Die $\overline{RAS}$-Leitung wird außerdem für die Auffrischung der dynamischen RAMs eingesetzt.

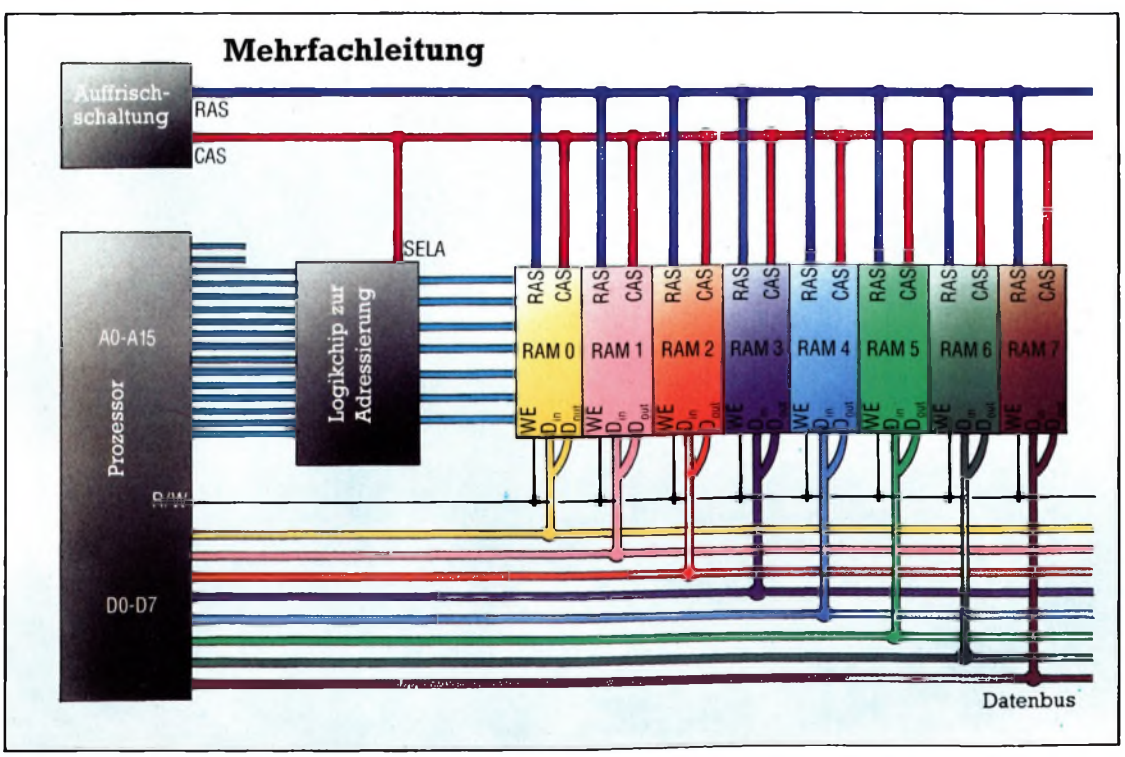
Auffrischung des Chips

Beim Auffrischungsvorgang werden alle Daten aus dem dynamischen RAM ausgelesen. Das Zurückschreiben stellt die Ladung wieder her. Da der Vorgang in unserem Beispiel zeilenweise abläuft, sind zur Auffrischung des Chips 128 Zyklen nötig. Die Auffrischung verlangsamt in vielen Fällen den Speicherzugriff des Prozessors, doch beträgt die Verzögerung selten mehr als fünf Prozent.

Die 14 Adressen eines 16-KByte-Speichers werden von unserem RAM in zwei Teilen zu je sieben Bits dargestellt. Das ermöglicht ein externer Logikchip, der die nieder- und höherwertigen Adreßleitungen mit dem CAS-Zeitsignal abstimmt. Unser Bild rechts oben zeigt, wie die niederwertigen Adreßleitungen (A0 bis A6) mit den Adreßkontakten des Chips verbunden sind, wenn CAS gesetzt ist. Ist CAS nicht gesetzt, sind die höherwertigen Adreßleitungen angeschlossen. Der RAM-Chip selbst enthält zwei Buffer, die ankommende Datensignale „einfrieren", so daß die Zeilen- und Spaltenadresse nacheinander ausgesandt werden können. Ein entsprechender Decoder (7 auf 128) setzt dann die hier erzeugten Adreßwerte zusammen.

Unser Bild rechts unten zeigt, wie ein dynamisches RAM von 16 KByte mit den Adreß-, Daten- und Schreib-/Lese-Leitungen des Prozessors verbunden ist.





Ein Speicher mit der Kapazität von 16 KByte läßt sich aus acht 4116 Chips aufbauen, die parallel geschaltet sind und sich die Leitungen für Adresse, R/W, RAS und CAS teilen. Bei dieser Anordnung ist jedes dynamische RAM mit einer Leitung des Datenbusses verbunden.

Die 14 Adreßleitungen, die jedes Bit eines 16-KBit-RAM-Chips ansprechen können, erzeugen je eine 7-Bit-Spalten- und Zeilenadresse. Zur Vereinfachung der Schaltung besitzt der Chip selbst nur sieben Adreßleitungen. Der kreisförmige Bildausschnitt zeigt, wie einfach die logische Schaltung des externen Logikchips aufgebaut ist, die zwischen den beiden Adreßhälften umschaltet.

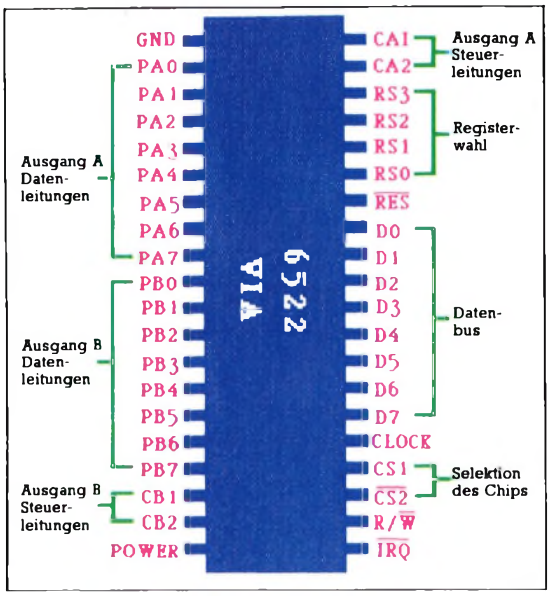
Der Torhüter

Ohne spezielle Ein- und Ausgabeeinheiten läßt sich keine Kommunikation mit der Außenwelt herstellen. Wir sehen uns den PIO-Chip genauer an, der über die Ein- und Ausgabe wacht.

Wir haben bereits untersucht, wie ein Prozessor arbeitet und mit ROM und RAM Daten austauscht. Unser Beispielsystem war jedoch nach außen geschlossen und daher nicht ganz realistisch. Die Grundkombination Speicher/Prozessor kann auf vielerlei Weise mit anderen Geräten kommunizieren. Eingaben gehen dabei über Tastatur, Joystick und andere.

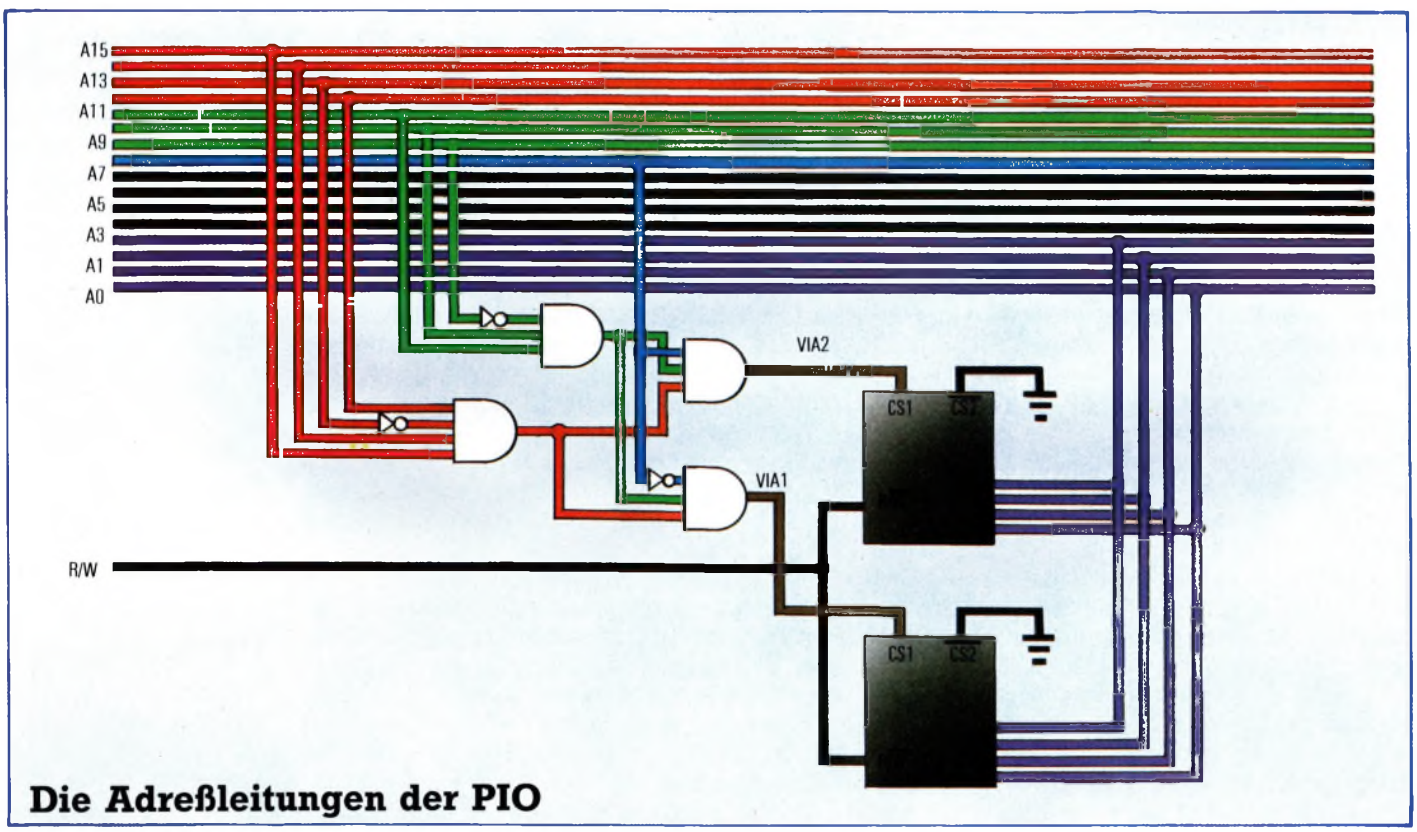
E/A-Geräte können nicht direkt an den Haupt-Datenbus angeschlossen werden, da die Daten dort nicht lange genug vorhanden sind, um von externen Geräten empfangen zu werden. Andererseits müssen eingehende Daten lange auf dem Datenbus bleiben, um vom Prozessor überhaupt verarbeitet werden zu können. Eine E/A-Schnittstelle muß Daten daher zumindest solange „festhalten", bis der E/A-Vorgang beendet ist, und ihn gleichzeitig synchronisieren (üblicherweise per Handshakeprozedur).

Die meisten E/A-Geräte bieten jedoch weit mehr als „Riegel" und Steuerleitungen. Wir müssen daher genauer auf die beiden E/A-Typen eingehen, mit denen die meisten Heimcomputer arbeiten. Der Schnittstellentyp wird hauptsächlich von der Methode bestimmt, mit der Peripheriegeräte und die Zentraleinheit Daten austauschen – seriell oder parallel. Es gibt daher auch zwei Gruppen von E/A-Geräten – PIOs und SIOs.

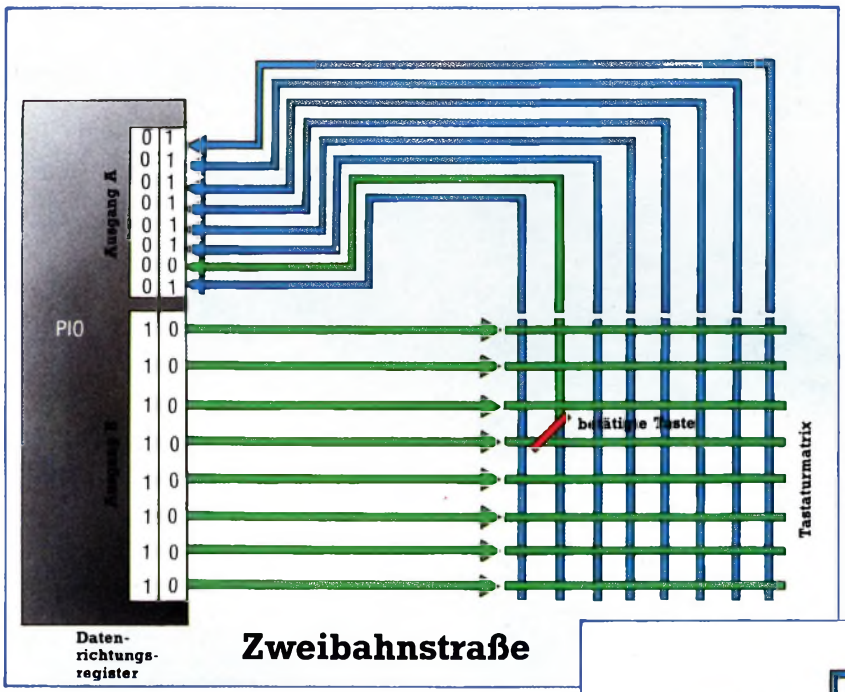


Der in den Acorn B eingebaute 6522 VIA-Chip ist ein typisches Beispiel für einen parallelen E/A-Chip – er besitzt zwei Acht-Bit-Ausgänge und Verbindungen zum Datenbus.

PIO-Chips werden über ihre internen Register programmiert. Um Zugang zu diesen Registern zu haben, muß der Prozessor sie in seinem Adreßbereich „finden" können. Unser Beispiel zeigt, wie zwei PIO-Chips in die Memory Map eingeblendet und über $DC00 und $DD00 angesprochen werden. Die oberen sieben Bits des Adreßbusses werden decodiert, die Leitung A8 wählt $DC00 oder $DD00 an.

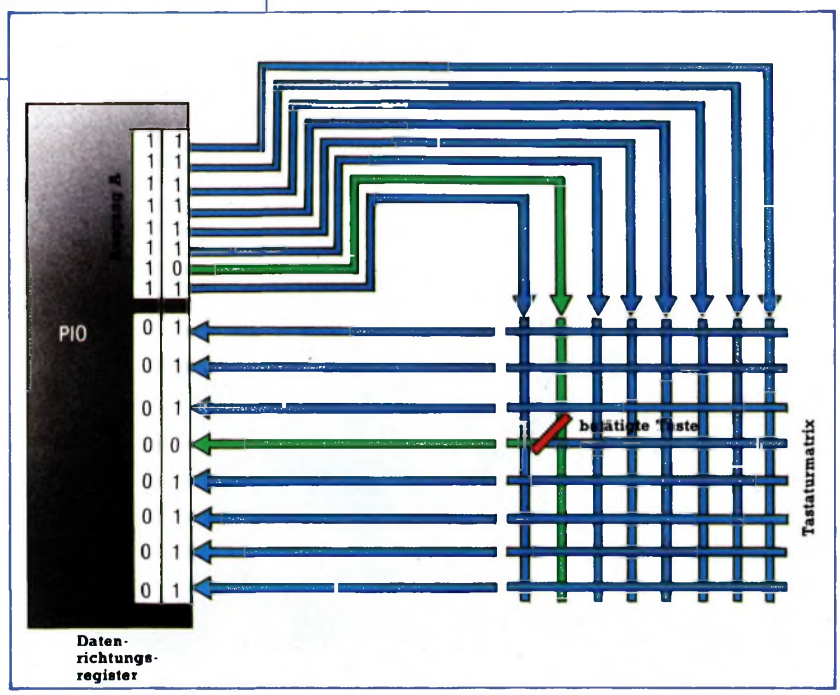


Die Adreßleitungen der PIO



Zweibahnstraße

Die Technik der Leitungsumkehrung ist eine von mehreren Möglichkeiten festzustellen, welche Taste einer Tastatur gedrückt wurde. Zunächst werden Nullen in die Matrixzeilen der Tastatur geladen. Ein Tastendruck setzt dann ein Bit dieses Registers auf Eins. Für die Spalten wiederholt sich der gleiche Vorgang in umgekehrter Richtung. Der so entstandene 16-Bit-Code kann das Zeichen der gedrückten Taste nun eindeutig identifizieren.



Die serielle Kommunikation wurde bereits ausführlich behandelt. Besonders die MIDI-Schnittstelle ist ein klassisches Beispiel für asynchrone Datenübertragung. Die Grundkomponenten einer derartigen Schnittstelle sind die Shift-Register, mit denen die parallel eingehenden Bits des Datenbusses in einen seriellen Datenstrom gewandelt werden und einkommende Daten paralleles Format erhalten. Auch testet die Schnittstelle die Prüfbits oder erzeugt sie.

Auf Heimcomputern ist der parallele PIO-Chip jedoch weit wichtiger. Der Name PIO wird nicht überall verwendet, VIA (Versatile Interface Adaptor), PIA (Peripheral Interface Adaptor) und CIA (Complex Interface Adaptor) bezeichnen alle die gleiche Einheit.

Heimcomputer mit voll ausgerüsteten Schnittstellen (etwa Acorn B und Commodore 64) besitzen zwei PIO-Chips. Einer davon steuert normalerweise die angeschlossenen E/A-Einheiten wie Tastatur und Joystick. Die zweite PIO läßt sich mit einem parallelen Drucker und einem User Port einsetzen und gibt Programmierern Zugang zum E/A-System.

Vielseitiger VIA-Chip

Unser erstes Bild zeigt den 6522 VIA-Chip des Acorn B. Mit seinen 40 Kontakten bietet er acht Verbindungen zum Datenbus und zwei Ausgänge im Acht-Bit-Format, die sich für Ein- und Ausgabe eignen. Die anderen Leitungen steuern Interrupts und Schreib/Lesevorgänge, registrieren Steuerbits und bedienen die Handshakeleitungen der E/A-Ausgänge.

Obwohl der PIO-Chip vom elektronischen Standpunkt aus gesehen extrem komplex ist, funktioniert er recht einfach. Der PIO-Chip eines typischen Acht-Bit-Micros hat zwei E/A-Ausgänge (A und B), bei denen sich die Datenrichtung der Bits mit Datenrichtungsregistern und separaten Handshakeleitungen individuell steuern läßt. Viele PIOs haben darüber hinaus noch Shift-Register, mit denen sie Daten über eine der Handshakeleitungen senden oder empfangen. Die Zeitsteuerung arbeitet dabei entweder mit einem der beiden internen Taktgeber oder mit den Impulsen eines externen Timers, die über eine andere Handshakeleitung hereinkommen.

Die einzelnen Bits eines Acht-Bit-Interrupt-Statusregisters zeigen an, ob von einer der vielen Quellen (Handshakesteuerleitungen, Timer, Shiftregister etc.) eine Interruptanforderung vorliegt. Bei programmierbaren Interrupts kann der Programmierer festlegen, welche der PIO-Interruptleitungen (falls nötig) die CPU unterbrechen soll.

Wie aber ist die PIO mit dem Grundsystem

Prozessor/Speicher verbunden? Die meisten Systeme mit E/A-Möglichkeiten verfügen über zwei Chips. Wenn die internen Register dieser beiden Chips bei $DC00 und $DD00 erscheinen sollen, müssen Daten an die PIO gesandt und von dort empfangen werden, das heißt, der Prozessor muß die internen PIO-Register wie Speicherstellen ansprechen und adressieren können. Damit jedoch PIO-Register im normalen Adreßbereich erscheinen, werden einige Adreßleitungen zur PIO geschaltet.

Über das Gatter

Die Abbildung der PIO-Adreßleitungen zeigt, wie die höchstwertigen acht Bits des Adreßbusses zu einer der beiden PIOs führen. Die oberen vier Bits werden über ein Gatter geleitet, das eine Eins erzeugt, wenn 1101 ($D) vorhanden ist. Die nächsten drei Bits lösen bei 110 eine Eins aus. Die Wahl zwischen $DC00 und $DD00 steuert das Adreßbit A8. Die untersten vier Bits sind direkt mit den Registerwahlkontakten der PIO verbunden. Auf diese Weise werden die internen Register der PIO 1 als Speicherstellen $DC00 bis $DC0F adressiert und die der PIO 2 von $DD00 bis $DD0F.

Eine PIO ist normalerweise für das System reserviert und stellt die Verbindung zu Tastatur und Joystick her. Es gibt mehrere Möglichkeiten, eine Tastatur mit 64 Tasten anzuschließen. Die einfachste arbeitet mit einer Leitungsmatrix zu je acht Zeilen und Spalten. Die Zeilenleitungen sind mit einem PIO-Ausgang verbunden, die Spaltenleitungen mit dem anderen. Ein Tastendruck verbindet die entsprechenden Zeilen- und Spaltenleitungen miteinander.

Die Technik der Leitungsumkehrung, mit der festgestellt wird, welche Taste gedrückt wurde, ist weitgehend softwaregesteuert. Sie arbeitet mit der Fähigkeit der PIO, die Ausgänge für Ein- und Ausgabe programmieren zu können. Dabei wird zuerst Ausgang B auf Ausgabe gesetzt und sein Datenregister mit Null geladen. Ausgang A ist auf Eingabe geschaltet, die Bits seines Datenregisters stehen auf Eins. Ein Tastendruck setzt nun das entsprechende Bit des Datenregisters A auf Null.

Der zweite Schritt kehrt die Datenrichtung der beiden Ausgänge um. Ausgang B ist nun bereit, Eingaben entgegenzunehmen. Die Bits seines Datenregisters stehen auf Eins – mit Ausnahme des Bits, das der Spaltenleitung der gedrückten Taste entspricht. Der jetzt in beiden Datenregistern gespeicherte 16-Bit-Code spricht die entsprechende Speicherstelle im ROM des Zeichensatzes an und ruft so den Acht-Bit-Code der gedrückten Taste ab.

Die beiden PIO-Chips können die Tastatur, einen parallelen Drucker und eine Anwenderschnittstelle mit dem Grundsystem verbinden. Im Bild unten ist zu sehen, daß die Tastatur einen PIO-Chip vollständig belegt. Einer der beiden Acht-Bit-Ausgänge der zweiten PIO dient als Druckerausgang, wobei ein Buffer zwischen Drucker und PIO inkorrekte Ausgaben abfängt. Der zweite Ausgang steht dann dem Anwender zur Verfügung.

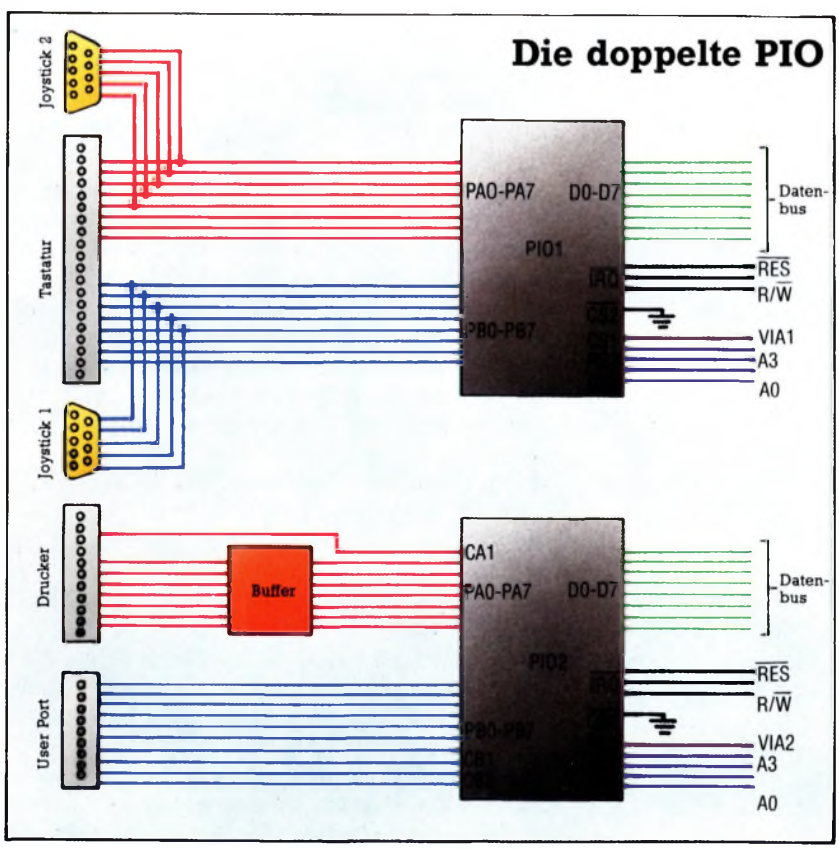
Das Bild zeigt, wie zwei PIO-Chips eine Schnittstelle bilden, die eine Tastatur, einen parallelen Drucker, Joysticks und den User Port mit dem Datenbus verbindet. Eine der Hauptaufgaben der Firmware oder Kernroutinen ist es, die PIO so zu programmieren, daß sie ein- und ausgehende Daten steuern kann. Für die meisten Anwendungen können Maschinencodeprogrammierer daher Kernroutinen verwenden, statt erst eigene Module entwickeln zu müssen. In einigen Fällen lohnt es sich jedoch, die PIO direkt anzusprechen – beispielsweise beim Einsatz des PIO-Timers oder des User Port.

Programmieren Sie die PIO

Der PIO-Chip besitzt mehrere interne Register, die der Hauptprozessor über das normale Adreßsystem ansprechen kann. Wenn diese Register mit bestimmten Werten geladen sind, werden in der PIO genau definierte Abläufe ausgelöst. Jede der acht Datenleitungen eines E/A-Ausgangs kann einzeln über das Datenrichtungsregister so programmiert werden, daß sie entweder als Eingangs- oder als Ausgangsleitung arbeitet. Auf diese Weise kann ein Ausgang mit einigen seiner Leitungen Eingaben entgegennehmen, während die anderen Leitungen Daten ausgeben.

Da E/A-Abläufe in fest definierten Zeitabschnitten ablaufen müssen, besitzt die PIO eigene Countdowntimer, die mit einem Wert geladen und bei jedem Taktsignal der Systemuhr dekrementiert werden. Das Auftreten eines „Underflow" (das heißt, der Zähler versucht, Null zu dekrementieren) löst ein Interruptsignal aus.

Synchrone Datenübertragung, bei der Daten in regelmäßigen Abständen empfangen oder gesendet werden, steuern meist die Timer der PIO.



Digitalmessung

Mit dieser Folge starten wir einen neuen Abschnitt im Selbstbau-Kurs, bei dem es um die Konstruktion und Arbeitsweise digitaler Vielfach-Meßgeräte gehen wird.

Normalerweise arbeitet ein Computer vollständig digital. Auch die Tastatur und der Videoausgang als „Schnittstelle" zum Menschen funktionieren nur durch Digitalschaltungen. Die Nutzbarkeit eines Computers jedoch wächst entsprechend seinen Fähigkeiten, mit der Außenwelt in Verbindung zu treten – einer Außenwelt, die größtenteils nicht dem Funktionskonzept der Maschine entspricht, sondern analog strukturiert ist: Spannungen sind immens variabel, ebenso wie Temperaturen, Höhen, Gewichte usw. Zum Erfassen von Daten oder zur Steuerung externer Geräte muß daher eine Umwandlung digitaler in analoge Werte und umgekehrt vorgenommen werden.

Ein Computer, der mit einem Temperaturfühler verbunden ist, muß beispielsweise Digitalzahlen erzeugen, die der gemessenen Temperatur entsprechen – andernfalls können die Meßdaten nicht vernünftig ausgewertet werden. Dazu wird ein „Analog-Digitalwandler (A/D-Wandler)" eingesetzt.

Das Gegenstück dazu, eine Einheit zur Verwandlung digitaler Werte in analoge Signale, wird immer dann eingesetzt, wenn ein digitales Gerät ein analoges Gerät steuern soll. Als Beispiel kann uns ein Rechner dienen, der für die Erzeugung von Musik programmiert wurde. Der Computer selbst erzeugt dabei nur einen digitalen (binären) Wert, der in entsprechende akustische (analoge) Signale umgesetzt werden muß. Das besorgt ein D/A-Wandler, der binäre Werte in Klänge verwandelt.

„Do it Yourselfer" gefordert

Ein Multimeter ist ein Gerät zum Messen von Widerständen (in Ohm), Stromstärke (in Ampere) oder elektrische Spannung (in Volt). In dieser Folge wollen wir ein digitales Multimeter bauen, das diese Werte sehr exakt feststellen und die Ergebnisse digital auf einem LED- oder LCD-Display darstellen kann. Ein solches Vorhaben würde die Fähigkeiten der meisten „Do-it-Yourselfer" übersteigen, wenn nicht in den letzten Jahren die Entwicklung hochintegrierter Chips hier vieles erleichtert hätte. Unser DVM (DigitalVoltMeter) soll so aufgebaut sein, daß es nicht nur allein funktionsfähig ist, sondern seine Meßergebnisse zur Weiterverarbeitung auch an den Computer übergibt.

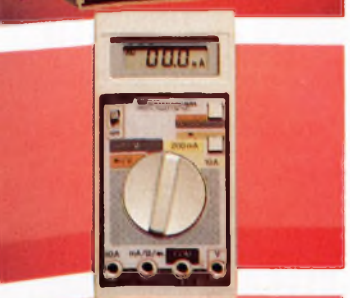
Bei einem herkömmlichen Voltmeter wird ein beweglich über einer Skala aufgehängter Zeiger verwendet, der durch seinen Ausschlag die gemessene Spannung, Stromstärke oder den Widerstand anzeigt. Ein Netzwerk aus Vor- und Nebenwiderständen sorgt dafür, daß die Anzeige den unterschiedlichen Meßbereichen (Volt, Ampere und Ohm) entspricht, obwohl die Grundfunktion des Meßwerkes immer die gleiche ist: Strom durchfließt eine Spule, die sich in einem Magnetfeld befindet. Die Stärke des fließenden Stromes bestimmt, wieweit sich die Spule aus ihrer Ruhelage entgegen der Kraft einer Feder bewegt. Ein mit der Spule verbundener Zeiger macht diese Abweichung aus der Ruhelage sichtbar.

Wenn eine unbekannte Spannung digital gemessen werden soll, gibt es sehr viel größere Schwierigkeiten. Bevor wir uns näher mit diesem Problem auseinandersetzen, müssen wir genauer analysieren, wie ein digitaler Wert in einen analogen umgewandelt wird:

Digital/Analogwandlung ist eigentlich ein recht einfacher Vorgang. Stellen Sie sich vor, Sie wollten ein Acht-Bit-Wort (eine Reihe von acht Binärziffern) in eine analoge Spannung

Multimeter-Marktübersicht
Hier sollen am Beispiel dreier Digitalmultimeter die Leistungsmerkmale der verschiedenen Geräteklassen dargestellt werden:

Lascar LMM 100	**Laborgerät**
Anzeige:	3 1/2-stellig LCD
Meßbereich:	25 Bereiche
Meßgenauigkeit:	0,1 % (DC Volt)
Versorgung:	Batterie
Weitere Merkmale:	Meßwerterhalt, Polarität, Batterieanzeige und Anzeige bei Bereichsüberschreitung.



Soar ME-531	**Meßgerät mit automatischer Bereichswahl**
Anzeige:	3 1/2-stellig LCD
Meßbereich:	Autom. Bereichswahl
Meßgenauigkeit:	0,8 % (DC Volt)
Versorgung:	Batterie
Weitere Merkmale:	Polarität, Überlastanzeige, Diodenprüfung

Precision Gold M-5010	**Einfaches Digitalmultimeter**
Anzeige:	3 1/2-stellig LCD
Meßbereich:	29 Bereiche
Meßgenauigkeit:	0,25 % (DC Volt)
Versorgung:	Batterie
Weitere Merkmale:	Durchgangsprüfer, Überlast- und Polaritätsanzeige, Diodenprüfung

zwischen 0 und 1 Volt umwandeln. Ein Acht-Bit-Wort kann einen Wert zwischen 0 und 255 darstellen (binär 00000000 bis 11111111). Der 1-Volt-Bereich läßt sich also in 256 Spannungsschritte oder Schritte von 0,0039 Volt auflösen.

Binäre Werte lassen sich „analogisieren", indem man sie zum Schalten von Spannungen nutzt. Die Zeichnungen stellen zwei Grundschaltungen für diese Funktion dar. In beiden Fällen werden die mechanischen Schalter elektronisch durch binäre Logikwerte bestätigt. Eins bedeutet dabei „geschlossen", Null „offen". Jede Binärziffer schließt einen der Schalter und vergrößert so den Stromfluß.

Die Analog/Digitalwandlung ist schwieriger. Wie bei der D/A-Wandlung bestimmt auch hier die Bitzahl die Genauigkeit (Auflösung) der Umwandlung oder Messung. Für unser Digitalvoltmeter ist eine Genauigkeit von acht Bit völlig ausreichend.

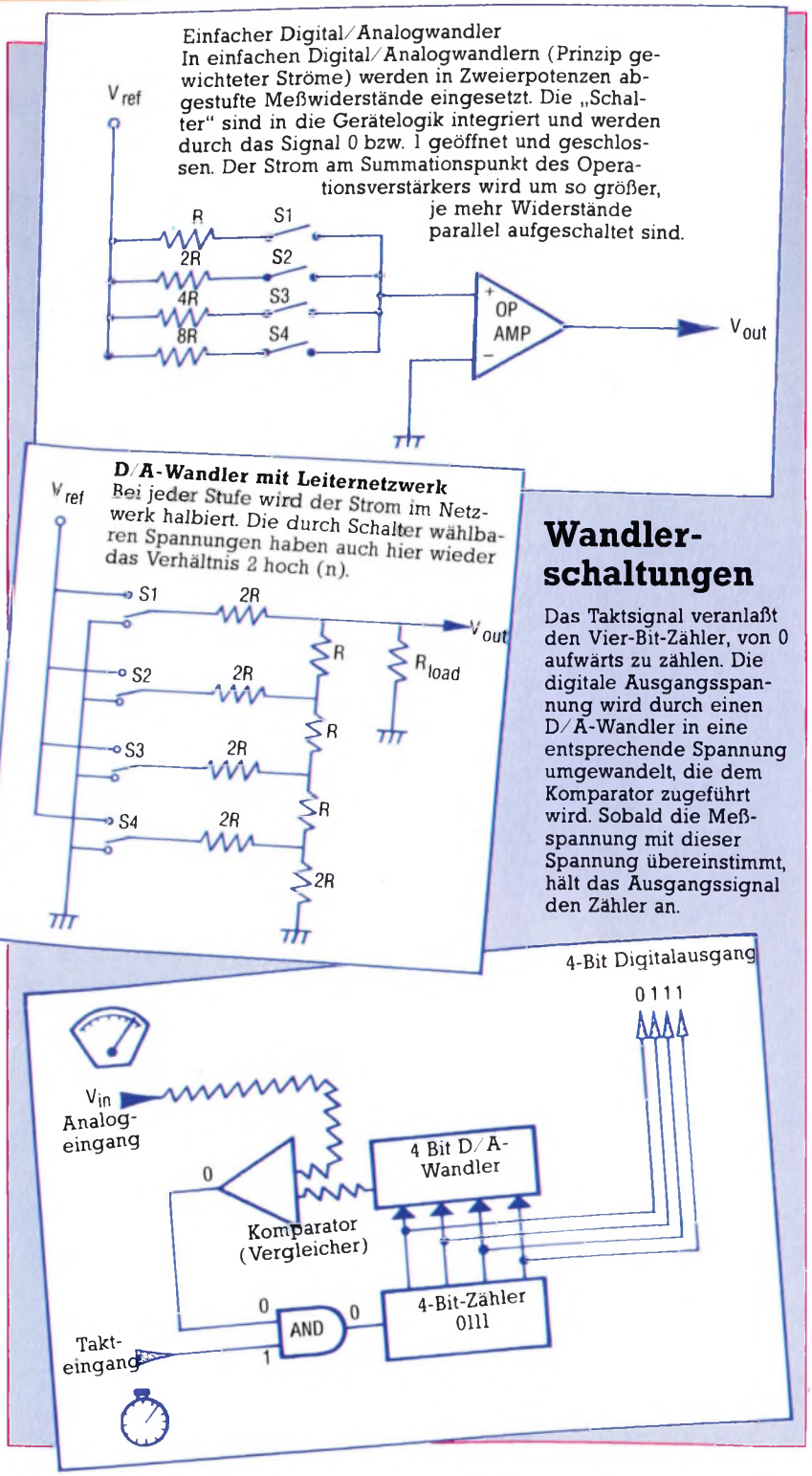
Die Schaltung soll in einem Basis-Arbeitsbereich von 0 bis 2 Volt funktionieren. Acht-Bit-Auflösung erlaubt dann Spannungsstufen von 0,0078 Volt. Der Meßbereich läßt sich durch ein einfaches Widerstandsnetzwerk auf 0 bis 20 Volt bzw. 0 bis 200 Volt erweitern.

Analog/Digitalwandlung ist erheblich schwieriger und damit auch teuerer als der umgekehrte Vorgang. Der Aufwand hängt davon ab, wie groß die Auflösung sein soll und wie schnell die Wandlung durchgeführt werden muß. Falls man Audio-Signale in einem Frequenzbereich von bis zu 15 kHz in digitale Signale umwandeln will, müssen in jeder Sekunde bis zu 30 000 Meßwerte genommen werden. Noch größer ist diese „Abtastrate" bei der Digitalisierung von Videosignalen. Zum Glück muß ein Digitalvoltmeter nur Gleich- und Wechselspannungen mit vergleichsweise niedriger Frequenz messen.

Integrierte Komparator-ICs

Die am häufigsten verwendete Methode ist der Vergleich einer bekannten Spannung (Referenzspannung) mit der unbekannten Spannung (Meßspannung). Integrierte Komparator-ICs sind für diesen Zweck ideal – bei entsprechender Beschaltung erzeugen sie bei Übereinstimmung der beiden angelegten Spannungen ein Ausgangssignal. Ein Binärzähler wird auf den Anfangswert 0 gesetzt und zählt von dort aufwärts, wobei er (durch die oben besprochene D/A-Technik) eine Spannung am Ausgang erzeugt, die irgendwann einmal zufällig mit der Meßspannung übereinstimmt. Das Ausgangssignal des Komparators hält dann den Zähler an, und die Meßspannung kann berechnet werden.

Es gibt zwar eine ganze Reihe integrierter Multimeter-Schaltungen, sie lassen sich aber im allgemeinen nur schwer mit einem Computer verbinden – ihre Ausgangsleitungen sind nur für eine LED- bzw. LCD-Anzeige geeignet.



Wir haben uns daher für den Einsatz des ICL 7135 von der Firma Analog Systems entschieden. Der binär-codierte dezimale Ausgangswert dieses ICs eignet sich nicht nur für die Ansteuerung eines Sieben-Segment-Displays, sondern auch für die Rechner-Kopplung.

Das IC hat einen sehr hohen Eingangswiderstand. Der Differenzeingang erlaubt die Messung sowohl positiver als auch negativer Spannungen. Außerdem gibt es einen automatischen Nullabgleich. Bereichsüber- und -unterschreitung sowie die Polarität der angelegten Spannung werden angezeigt.

Gespeicherter Schirm

Bei der Betrachtung der Komponenten eines Computers untersuchen wir zwei unterschiedliche Arten von Video-Chips und sehen, wie Binärmuster in Bildschirminhalte umgewandelt werden.

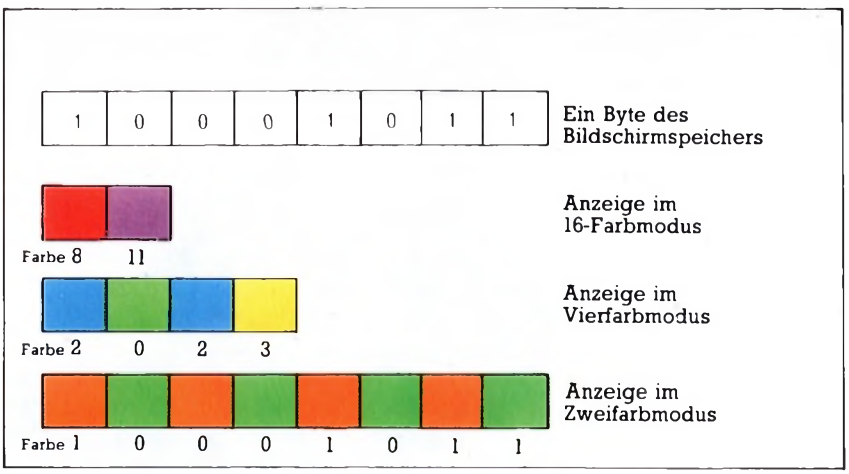
Qualität und Vielseitigkeit der Bildschirmdarstellung sind wichtige Verkaufsargumente für Computer. Wieviel Farben sind möglich? Wieviel Zeichen lassen sich in einer Zeile darstellen? Lassen sich Sprites steuern? Viele dieser Fragen beantwortet der Video-Chip des Gerätes.

Ein Video-Chip (auch CRTC = „Cathode Ray Tube Controller" – Bildschirmsteuerung – genannt) führt zunächst einmal Grundfunktionen aus: Er nimmt die Bytes der im RAM gespeicherten Anzeigedaten und bringt sie als Formen und Farben auf den Bildschirm. Unterschiede zwischen Videochips beziehen sich hauptsächlich auf die Technik, mit der sie Daten in Bildinformationen umwandeln.

Die erste Art von Video Chip hält Anzeige und Farbdaten gemeinsam im Speicher und macht damit die Umwandlung in ein Bild recht einfach. Mit dieser Methode arbeiten unter anderem Schneider-Computer. Chips dieser Art unterstützen üblicherweise eine ganze Palette unterschiedlicher Anzeigearten und können Zeichengröße, Farbenzahl etc. beeinflussen. Unser erstes Bild zeigt, wie unterschiedliche Anzeigenarten ein einzelnes Byte interpretieren. Obwohl die Arbeit des Video-Chips durch diese Technik sehr vereinfacht wird, und auch das Mischen von Text und hochauflösender Grafik möglich ist, belegt diese Methode mit seinem vollgepackten Bildschirmspeicher jedoch einen großen RAM-Bereich. Eine Anzeige mit 160 mal 256 Pixel und 16 Farben braucht beispielsweise 20 KBytes.

Grafiken aus Zeichen

Als Speicherchips noch teuer waren, ließ sich diese Methode nicht selbstverständlich einsetzen. So entstand eine alternative Anzeigetechnik, die auf dem Commodore 64 heute noch Anwendung findet. Statt jedes Pixel mit seiner Farbinformation im Speicher zu halten, baut dieser Video Chip die Grafik aus einzelnen Zeichen auf. Dabei entspricht die Position eines Zeichens je einem Speicherbyte. Jedes Byte enthält einen Zeichencode, über den der Video Chip eine ROM-Tabelle mit der Bildschirmmatrix von acht mal acht Pixel abruft.



Die Videosteuerung des Acorn B oder der Schneider-Computer bringt alle Bildschirminformation (Pixel AN/AUS und Farbdaten) zusammen im Speicher unter. Beide Computer besitzen mehrere Anzeigearten, die die gespeicherten Bitmuster auf unterschiedliche Weise interpretieren. In einem 16-Farben-Modus sind vier Bits für die Speicherung des Farbcodes nötig. Ein Byte kann daher nur je zwei Pixel darstellen. Im Zwei- oder Vierfarbmodus werden nur ein (bzw. zwei) Bits für den Farbcode eines Pixels benötigt. In diesen Darstellungsarten kann ein Byte vier bzw. acht Pixel speichern.

Eine Zeichenanzeige mit 40 mal 25 Zeichen belegt hier weniger als ein KByte. Die Farbdaten dieses Systems sind in weiteren 1000 Bytes gespeichert, die den Farbcode für die entsprechenden Zeichenbytes liefern.

Auch das Anzeigesystem des Schneider CPC greift auf im ROM gespeicherte Zeichendefinitionen zu, spricht sie aber vom Betriebssystem aus an, wenn Zeichen zum Bildschirmspeicher geschickt werden sollen.

Bildschirmanzeigen, die mit der zweiten Methode arbeiten, haben normalerweise einen zweiten, hochauflösenden Modus, bei dem die Speicherbits den Pixeln des Bildes direkt entsprechen. Diese Art der Grafikaufbereitung wird „Bit-Mapping" genannt. Die Farbinformation ist in diesem Fall nicht wie im ersten System in den Anzeigebits abgelegt, sondern in einem separaten Speicherbereich untergebracht. Das heißt, jedes Byte des zweiten RAM-Bereichs von 1000 Bytes enthält die Farbdaten für ein Feld von acht mal acht Pixeln.

Auf dem Commodore 64 ist damit zwar hochauflösende Grafik möglich, doch lassen sich Text und Grafik nicht mischen – es sei denn, Sie entwickeln eigene Routinen, die die Zeichendefinitionen in die Bit-Map schreiben. Dieser Ansatz wurde interessanterweise vom Spectrum übernommen, der seinen Bildschirm als einfache Bit-Map speichert und mit einem separaten Farb-RAM die Farben der einzelnen Zeichenfelder (acht mal acht Pixel) steuert.

Auf dem Spectrum lassen sich Text und hochauflösende Grafik problemlos mischen, da das Betriebssystem die Zeichendefinitionen in die Bit-Map schreibt.

Auch die Sprite-Grafik wird vom Video-Chip gesteuert. Normalerweise werden Sprites als Bitmuster im RAM definiert und ihre Eigenschaften (Farbe, Größe etc.) über die internen Register des Video-Chips gesteuert. Da ein einziger Chip außer der normalen Grafik auch die Sprites darstellen muß, lassen sich Video-Chips wie der VIC des Commodore 64 auf die Erzeugung von Interrupts programmieren, wenn Sprites mit beispielsweise anderen Sprites zusammenstoßen.

Verschiedenes Scrollen

Das Scrollen des Bildschirms führen die beiden hier beschriebenen Video-Chips auf unterschiedliche Weise aus. Der zweite Chiptyp steuert das Scrollen per Software. Sein Bildschirmspeicher enthält nur wenig Bytes, die sich schnell mit Maschinencoderoutinen umstellen lassen. Beim ersten Chip mit seiner großen Zahl von Bildschirmbytes wäre das software-gesteuerte Scrollen viel zu langsam. Diese Chips haben daher ein Spezialregister für die Anfangsadresse des Bildschirmspeichers und steuern das Scrollen über eine Änderung dieses Registerinhalts. Bei umsichtigem Einsatz dieser Methode kann die Anzeige in alle vier Richtungen gescrollt werden.

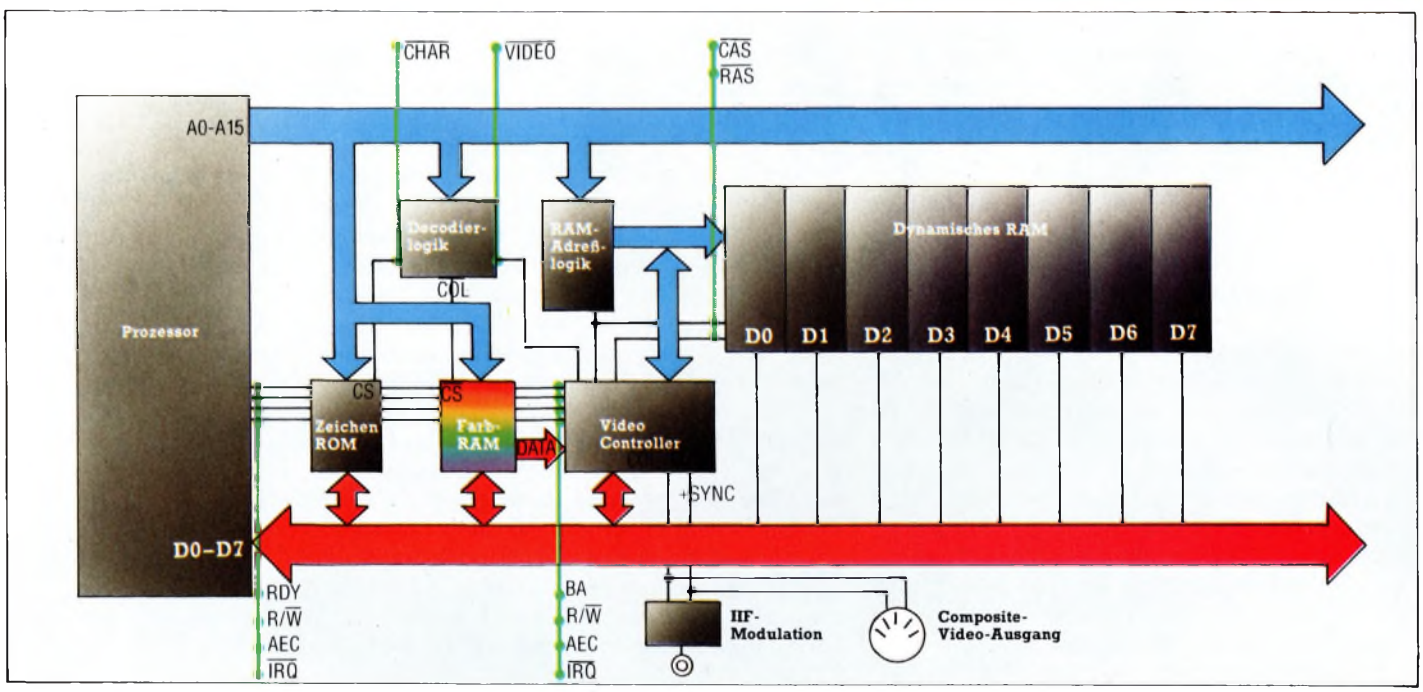
Der in den Acorn B eingebaute Video-Chip hat weit weniger zu tun, als beispielsweise der VIC-Chip des Commodore 64. Er kann seine Speicherzugriffe in die Phase des Taktzyklus verlegen, in dem der Prozessor den Speicher nicht anspricht. Damit verlangsamt der Video-Chip die Abläufe des Computers nicht. Außer einem Video-Chip besitzt der Acorn B aber noch eine ULA, die das gesamte System mit Zeitsignalen versorgt, die Beziehung zwischen logischen und physischen Farben berechnet und die RGB-Ausgabe liefert.

Im Gegensatz dazu sind mit dem VIC-Chip mehrere Speicherzugriffe nötig. Einige davon – beispielsweise die RAM-Auffrischung und das Abrufen von Zeichendaten haben keine Auswirkungen auf die Prozessorabläufe, da sie in Pausen der Systemuhr ausgeführt werden.

Immer im Bild

Das Bild zeigt den Anschluß des VIC-Chip an das System des Commodore 64. Der VIC ist direkt mit dem Datenbus verbunden und arbeitet mit den gleichen Adreßleitungen (im Multiplexverfahren) wie das dynamische RAM, das er ebenfalls über die Steuerleitungen CAS und RAS auffrischt. Einige Abläufe des Video-Chips lassen sich ohne Unterbrechung des Prozessors ausführen. Weiterhin stellt die Steuerung des Adreßbusses (AEC = Address Bus Enable Control) sicher, daß die Adreßbustreiber des Prozessors während der Phase der Systemuhr abgeschaltet sind, in der der Video-Chip den Speicher anspricht.

Der Zugriff auf die 1000 Byte des Bildschirmspeichers und die Spritebereiche muß jedoch öfter geschehen, als in jeder zweiten Phase. Es ist daher notwendig, den Prozessor auszuschalten, wenn der Video-Chip für den Abruf seiner Daten Speicherzyklen braucht. Das Ausschalten geschieht über eine Verbindung der VIC-Leitung BA (‚Bus Available' = Bus verfügbar) mit dem READY-Kontakt des Prozessors. Ihm stehen drei weitere Zyklen zur Verfügung, um einen bereits laufenden Speicherzugriff abzuschließen. Im vierten Taktzyklus des Prozessorzugriffs bleibt AEC jedoch auf „Low" und gibt dem VIC so die Möglichkeit, den Speicher anzusprechen.



Bits im Wandel

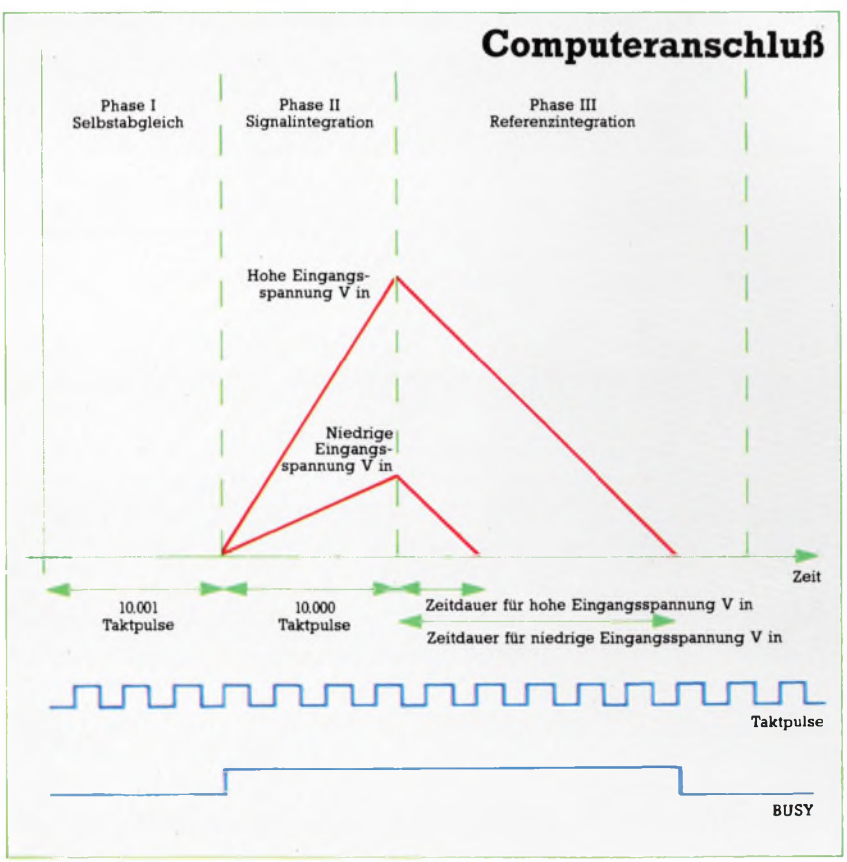
Nach Erläuterung der Grundlagen einer D/A- oder A/D-Wandlung und dem Entwurf unseres Digitalmultimeters untersuchen wir jetzt die am weitesten verbreiteten Wandlertechniken. Den integrierten Wandler 7135 wollen wir etwas detaillierter beschreiben – er soll in unserer Konstruktion verwendet werden.

Während die Zeit für die Phasen I und II festgelegt ist, ist die Dauer der dritten Phase proportional zur Eingangsspannung V in. Während dieser Phase wird die vorher integrierte Eingangsspannung gleichmäßig auf Null reduziert: Das dauert um so länger, je größer die Spannung anfangs war. Man kann deshalb unser Gerät sehr einfach mit einem Computer koppeln.
Der BUSY-Ausgang des 7135 geht zu Beginn der Signal-Integration auf „High" und bleibt dort bis einen Puls nach dem Nulldurchgang des Integrators. Ein logisches AND zwischen den Taktpulsen und dem BUSY-Signal ergibt die Anzahl gültiger Pulse. Diese Pulse können über ein serielles Interface an den Rechner weitergegeben werden.
Zieht man hiervon die 10 001 Pulse der Signal-Integration und den einen am Ende der Referenz-Integration ab, ist die verbleibende Zahl der Eingangsspannung direkt proportional.

Die meisten D/A-Wandler arbeiten entweder nach dem Wägeverfahren oder mit integrierender Wandlung. Das gilt sowohl für auf ICs zusammengefaßte Schaltungen als auch für einen „diskreten" Aufbau, also ein Gerät aus Einzelbauteilen. Das IC 7135 ist ein integrierter Wandler. Bevor wir seine Funktion genauer untersuchen, wollen wir kurz die Grundlagen beider Methoden beschreiben.

Beim Wägeverfahren benutzt man einen D/A-Wandler in einem Rückkopplungskreis zusammen mit einem Komparator (Vergleicher). Die Ausgangsspannung des D/A-Wandlers wird Bit für Bit verändert, wobei man mit dem höchstwertigen Bit (MSB) anfängt und sich zum niederwertigen (LSB) vorarbeitet. Zu Beginn werden alle Bits auf Eins gesetzt. Solange die Eingangsspannung noch größer als die Ausgangsspannung des D/A-Wandlers ist, bleibt das untersuchte Bit auf Eins, andernfalls wird es zurückgesetzt. Dann geht es mit der nächsten Stelle weiter. Sind alle Bits getestet (meist 8, 12 oder 16), erzeugen die auf 1 stehenden (gesetzten) Bits am D/A-Wandler eine Spannung, die der am Analogeingang anliegenden exakt gleicht.

Dazu braucht eine Acht-Bit-Wandlung acht Vergleichsoperationen, eine 16-Bit-Wandlung 16 usw. Das Wägeverfahren arbeitet sehr schnell, 100 000 Wandlungen in der Sekunde oder mehr sind keine Seltenheit. Damit ist es ideal für Anwendungen in der Audio- Video- oder Radartechnik, wo eine Vielzahl analoger Daten in kürzester Zeit auf Digitalform gebracht werden müssen. Nachteilig ist, daß Wägekonverter sehr teuer sind und eine überaus komplizierte Beschaltung benötigen.



Sample and hold

Wenn, wie etwa bei Digitalmultimetern, nicht die Geschwindigkeit im Vordergrund steht, entscheidet man sich üblicherweise für einen integrierten Wandler. Damit können Wandlungsraten von einigen Messungen pro Sekunde erreicht werden. Das reicht fast immer – vorausgesetzt, man möchte nicht gerade hochfrequente Spannungsverläufe untersuchen.

Der Ausgang eines integrierten Wandlers gibt den Mittelwert der analogen Eingangsspannung innerhalb einer fest gewählten Zeit an. Anders als beim Wägeverfahren muß die analoge Eingangsspannung nicht zwischengespeichert werden (Sample and hold). Der zeitliche Ablauf der Wandlung wird durch die Pulse eines Taktgenerators gesteuert.

Die Vorzüge der in unseren Multimeter eingesetzten integrierten Wandlung liegt in der hohen Genauigkeit, dem sehr guten Rauschverhalten und einem Aufbau, der – abgesehen von der Schaltung zur Erzeugung einer Referenzspannung – ohne „kritische" Bauteile auskommt. Außerdem fällt die „Sample and hold"-Komponente der oben beschriebenen Wandlungsmethode fort, was sich auch durch geringere Kosten angenehm bemerkbar macht.

Bei einem typischen Wandler dieses Typs wird die Konversion in drei Phasen vorgenommen: Auf den Selbstabgleich folgt die Signal-Integration und Referenz-Integration. Durch diese Technik wird eine relativ große Unempfindlichkeit gegenüber hochfrequenten Störun-

gen und Eingangsrauschen erreicht.

Neben einem stabilen Taktgenerator braucht man dafür nur eine genaue Referenzspannung, die aber mit handelsüblichen Zener-Dioden recht einfach erzeugt werden kann. Diese Dioden begrenzen die höchstzulässige Eingangsspannung.

Die meisten integrierten Wandler arbeiten nach dem sogenannten „Dual-Slope"-Verfahren, dessen Ablauf so aussieht:

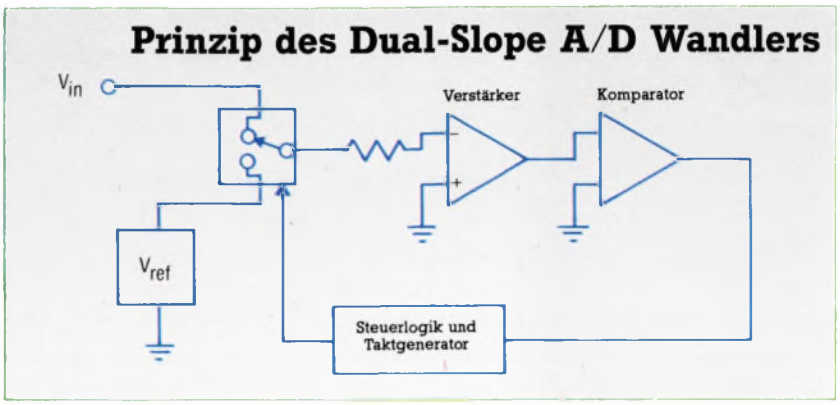
Phase 1/Selbstabgleich: Der Eingang wird auf Masse gelegt, die Fehlerinformation über die analogen Eingangskomponenten wird in einem Kondensator gespeichert.

Phase 2/Signal-Integration: Das Eingangssignal wird über eine bestimmte Anzahl von Taktpulsen integriert – für eine Wandlung mit 4 1/2 Stellen sind 10 000 üblich. Nach der Integration ist die erzeugte Spannung dem Eingangssignal direkt proportional.

Phase 3/Referenz-Integration: Zu Beginn dieser Phase wird der Eingang des Integrators an die Referenzspannung gelegt. Die Zahl der Pulse, die zwischen dem Anlegen der Referenzspannung und dem Nulldurchgang des Integrators gezählt wird, ist dann der Eingangsspannung direkt proportional.

Dual-Slope A/D-Wandler sind sehr exakt, weil das Meßergebnis nur von der Genauigkeit der Referenzspannung und der Stabilität des Taktgebers abhängt. Die verwendete Zenerdiode sollte deshalb sorgfältig ausgewählt werden, die Frequenz des Taktgenerators ist dagegen relativ beliebig, auch das Tastverhältnis ist nicht wichtig. Wir haben uns daher für das Timer IC 555 entschieden, dessen Frequenzstabilität für diese Anwendung ausreicht.

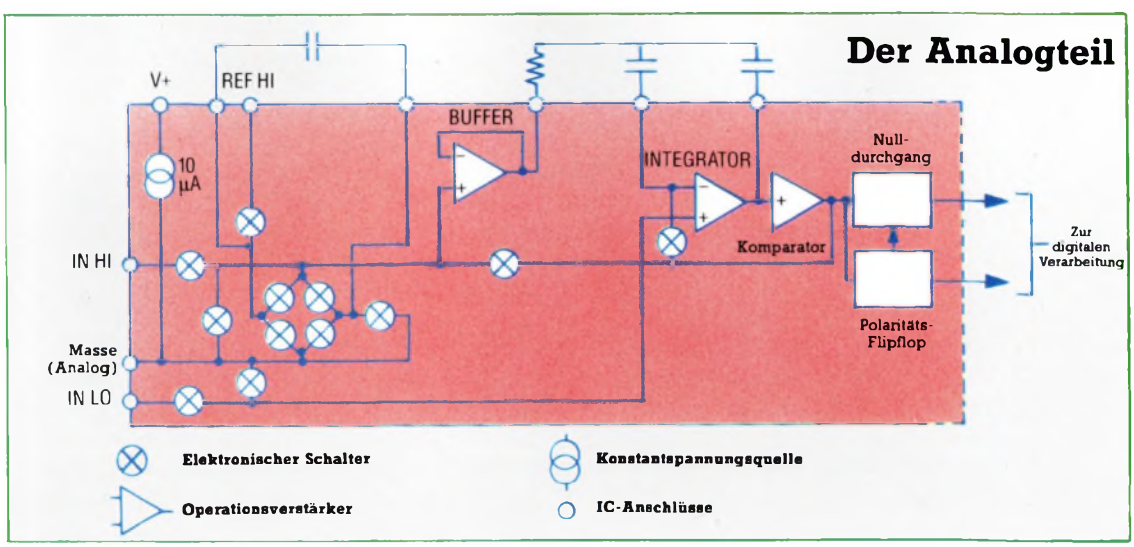
Die Stabilität anderer Komponenten, etwa des Integrationskondensators, ist ohne Bedeutung, solange sich sein Wert nicht innerhalb einer Wandlung ändert. Beim Wägeverfahren hängt die Exaktheit des Ergebnisses dagegen sehr stark von der Genauigkeit des Widerstands-Netzwerks ab. Da die Genauigkeit der Uhr leicht auf Eins zu eine Million gebracht werden kann, sind Dual-Slope-Wandler dem Wägeverfahren deutlich überlegen – jedenfalls, solange auf hohe Wandlungsgeschwindigkeit kein Wert gelegt wird.



Der Schaltplan zeigt den analogen Teil des 7135-ICs. Die meisten Symbole kennen Sie bereits, der Doppelkreis und die durchkreuzten Kreise sind Ihnen aber vielleicht noch neu. Der Doppelkreis steht für eine Konstantspannungsquelle, die durchkreuzten Kreise stellen elektronische Schalter dar.
Ein integrierender Dual-Slope A/D-Wandler arbeitet in drei Phasen: Selbstabgleich, Signal-Integration und Referenz-Integration. Während der Signal-Integration wird das Eingangssignal -Vin- und während der Referenz-Integration die Referenzspannung -Vref- eingeschaltet. Der Ablauf wird durch einen externen Taktgeber gesteuert.

Es ist wichtig, das Zeitverhalten in den drei Phasen der Wandlung – Selbstabgleich, Signal-Integration und Referenz-Integration – etwas genauer zu betrachten. Die Anzahl der Taktpulse in den ersten beiden Phasen ist festgelegt, nämlich 10001 Pulse für den Selbstabgleich und 10000 Pulse für die Signal-Integration. Die Anzahl der Pulse in der dritten Phase dagegen ist variabel. Es werden bis zu einer Obergrenze von 20001 so viele Pulse gezählt, bis der Ausgang des Integrators einen Nulldurchgang macht. Daher können die Ergebnisse mit relativ unkomplizierter Technik über ein serielles Interface an einen Computer weitergegeben werden.

Die Frequenz des Taktgenerators spielt innerhalb weiter Grenzen keine Rolle. Sie kann bei 5 kHz liegen (die Wandlung dauert dann etwa zehn Sekunden), oder bei 1 MHz. Bei zu niedrigen Frequenzen treten allerdings Fehler durch die Leckströme des Integrationskondensators auf, bei sehr hohen Frequenzen muß der Kondensator dagegen durch ein Widerstandsnetzwerk kompensiert werden. Am wenigsten Probleme gibt es bei Frequenzen zwischen 100 und 160 kHz.



Das Diagramm zeigt den analogen Flußplan des 7135 A/D-Wandlers. Bauteile im schattierten Gebiet befinden sich auf dem Chip, die anderen Komponenten müssen extern dazugeschaltet werden. Der Analogteil des 7135 versorgt den digitalen Teil mit Informationen über den Nulldurchgang und die Polarität. Zusammen mit den Signalen des externen Taktgebers kann der Digitalteil des ICs daraus die gewünschten Daten berechnen und über den BCD-Anschluß des Chips ausgeben.

Stück für Stück

Im letzten Abschnitt haben wir uns mit den Prinzipien der Analog/Digital-(A/D) bzw. der Digital/Analogwandlung (D/A) befaßt. In dieser Folge geben wir einen Gesamtüberblick – alle Komponenten unseres Meßinstruments werden einzeln vorgestellt und erklärt.

Die Schaltung des Digitalvoltmeters besteht aus sieben Abschnitten bzw. Blocks (siehe Zeichnung). Wir wollen diese Komponenten im einzelnen genauer beschreiben.

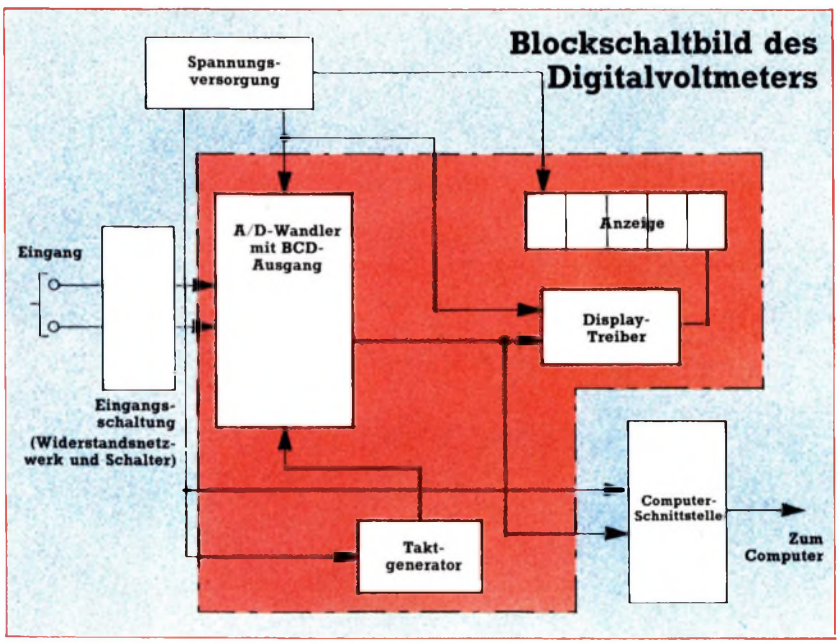
● **Eingangsabschwächer und Schalter:** Das Eingangssignal wird über Meßleitungen einem Widerstands- und Schalternetzwerk zugeleitet, bevor es zur eigentlichen Meßschaltung gelangt. Der Umfang dieser vorgeschalteten Einheit hängt davon ab, wie groß der Meßbereich des Instruments sein soll (etwa von Microvolt bis Kilovolt). Außerdem ist dafür entscheidend, welche elektrischen Größen gemessen werden sollen (Gleichspannung, Wechselspannung, Gleichströme, Widerstände usw.). Der Aufbau dieser Eingangsschaltung wird später noch genauer dargelegt.

● **Die Spannungsversorgung:** Dazu ist wenig zu sagen – es sind nur die Spannungen +5 Volt und –5 Volt nötig, wobei die Belastung recht klein ist. Wir können für diesen Zweck zei Festspannungsregler verwenden und entweder die Netzspannung oder Batterien einsetzen. Bis auf die Eingangsschaltung, die nur passive Bauelemente enthält, sind alle Systemkomponenten mit der Spannungsversorgung verbunden. Beim Aufbau muß sehr sorgfältig verdrahtet werden, um unerwünschte Masseschleifen zu vermeiden.

● **Die Taktgenerator-Schaltung:** Der A/D-Wandler 7135 braucht – wie fast alle anderen Wandler auch – externe Takt-Signale. Das Taktsignal kann sehr einfach mit rückgekoppelten TTL-Bausteinen (TTL=Transistor-Transistor-Logik) erzeugt werden. Wir haben uns für das vielseitige Timer-IC 555 entschieden. Der interne Aufbau ist recht kompliziert, zur Außenbeschaltung sind aber nur zwei Widerstände und ein Kondensator nötig.

● **Die Computer-Schnittstelle:** Diese Komponente ist ein Extra, auf das auch verzichtet werden kann. Unsere Schaltung ist als unabhängiges Gerät konzipiert und soll eine Alternative zu den teuren fertigen Digitalvoltmetern sein. Falls Ihr Computer aber Meßwerte aus seiner Umgebung verarbeiten soll – beispielsweise Spannungen, Widerstände, Temperaturen – können Sie die Schnittstelle nachrüsten.

● **Der A/D-Wandler:** Der Wandler ist „Herz" des Digitalvoltmeters. Er ist aus einem einzigen IC aufgebaut, das einen Standard-Meßbereich von zwei Volt hat (genauer gesagt 1,9999 Volt).



Das Diagramm verdeutlicht das Zusammenwirken der einzelnen Komponenten unseres Digital-Multimeters. Der rote Bereich des Blockdiagramms entspricht dem auf der nächsten Seite detailliert dargestellten Teil-Schaltplan.

Höhere Spannungen müssen von der Eingangsschaltung erst reduziert werden – am Wandler darf keine Spannung über zwei Volt liegen.

Vorteile beim Chip

Der verwendete Chip (IC 7135) ist anderen A/D-Konvertern in mehrfacher Hinsicht überlegen. Er kann die Eingangsspannung im BCD-Code (Binär Codierte Dezimalwerte) angeben und ist dadurch sehr viel leichter mit einem Computer zu verbinden als Wandlertypen, die für die Ansteuerung von Sieben-Segment-Anzeigen vorgesehen sind. Aber der Chip kann noch mehr: Das IC 7135 arbeitet sehr genau und liefert im Zwei-Volt-Bereich vier Dezimalstellen. Ein automatischer Nullabgleich sorgt dafür, daß ohne angeschlossene Spannung wirklich Null in der Anzeige steht.

Das IC 7135 verfügt über einen Ausweg für die Meldung einer Meßbereichs-Unterschreitung. In unserer einfachen Schaltung können wir ihn zur Ansteuerung einer LED gebrauchen; sie macht uns darauf aufmerksam, daß wir in einen kleineren Meßbereich umschalten können. In hochentwickelten Digitalvoltmetern werden die Meldeleitungen der Meßbereichs-über- und Unterschreitung für die selbsttätige Bereichswahl genutzt.

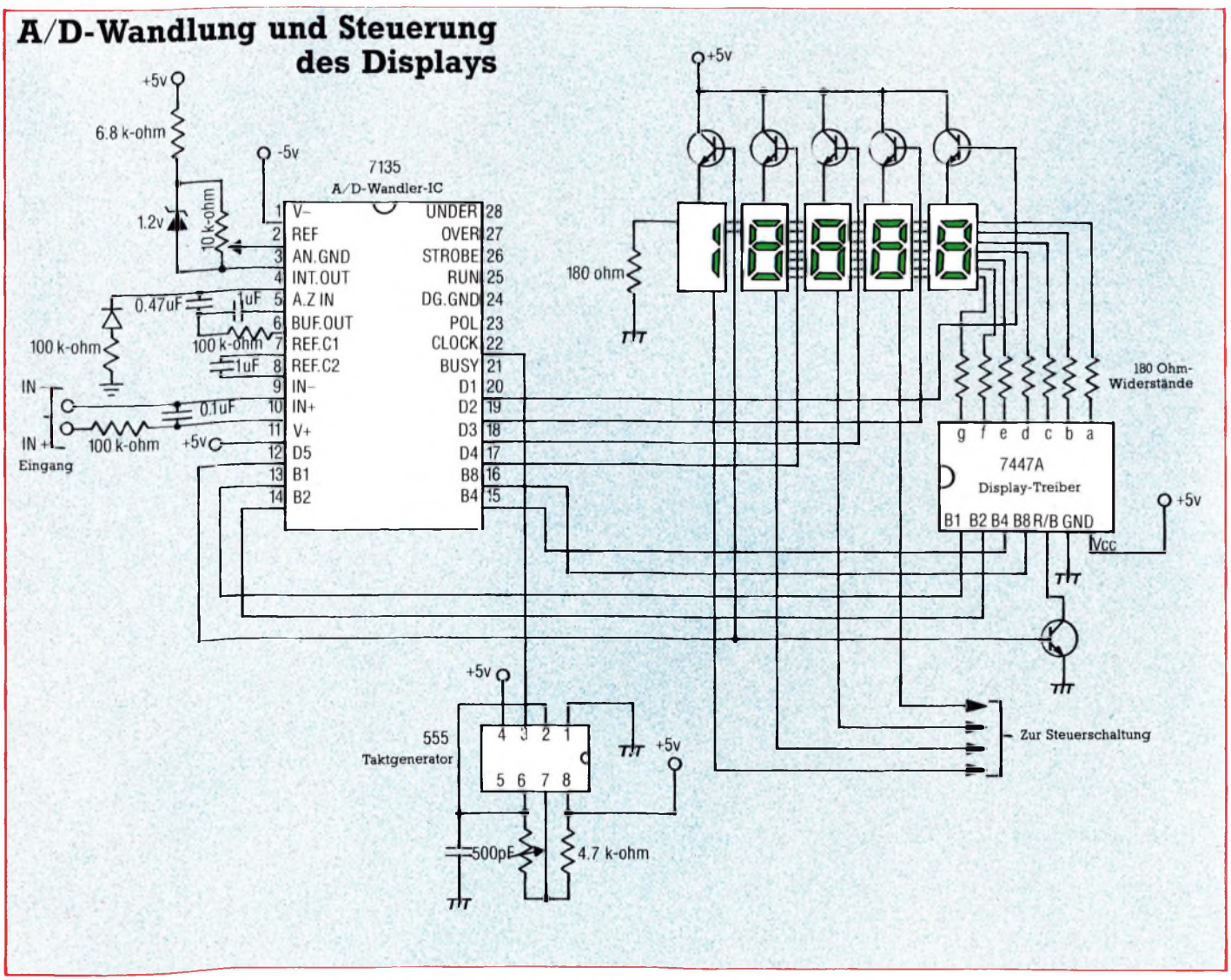
Die Grundlage unseres Digitalvoltmeters ist das A/D-Wandler-IC 7135. Das Bauteil wird extern von einem 555-Chip mit Taktsignalen versorgt. Dieser Baustein ist unten im Schaltbild dargestellt. Der 7135 gibt Signale im BCD-Code aus, die von einem 7447A-IC für die Darstellung über Sieben-Segment-Anzeigen aufbereitet werden. Durch fünf Schalttransistoren können alle fünf Anzeigefelder im Multiplex-Verfahren durch ein einziges Treiber-IC gesteuert werden. Da die Schaltung in dieser Form noch nicht vollständig ist, sollten Sie mit dem Bau aber noch nicht beginnen.

Die Ausgangsspannungen des ICs sind direkt TTL-kompatibel – das erleichtert die Verbindung zum Computer. Dazu liefert die Schaltung auch einige Signale, mit denen der Anschluß am Rechner einfacher wird. Dazu zählt eine STROBE-Leitung, die den Datentransfer zu externen Latches erleichtert (Latch=Zwischenspeicher für Daten, der ohne ständige Eingabe auskommt). Durch eine RUN/HOLD-Eingabeleitung kann vom „Einfrieren" des aktuellen Meßwertes auf freilaufenden (unabhängigen) Betrieb des Multimeters umgeschaltet werden. Die BUSY-Leitung meldet dem Rechner, ob Daten gültig sind oder nicht: Der Zustand der BUSY-Leitung gibt Auskunft darüber, ob die Daten auf den BCD-Leitungen ausgewertet werden können oder sich gerade im nicht definierten Zustand befinden.

● **Display-Treiber:** Der 7135-Wandler liefert am Ausgang ein BCD-Signal, das für die direkte Steuerung einer Anzeige nicht geeignet ist. Zum Glück gibt es einen preiswerten IC-Baustein, der ohne zusätzliche Beschaltung den BCD-Code für die Displaysteuerung umsetzt: Das IC 7447A liefert Signale, mit denen eine einzelne Sieben-Segment-Anzeige versorgt werden kann. Bei einer fünfstelligen Anzeige müssen wir im Multiplexverfahren arbeiten. Die vier BCD-Signale werden allen Anzeigen zugeleitet, sind aber jeweils nur für eine der Sieben-Segment-Anzeigen gültig. Welche das ist, wird durch Signalleitungen vom IC 7135 angegeben. Die Signale für die jeweils gültige Dezimalstelle werden verwandt, um über Transistoren nur je eins der Anzeigefelder zu aktivieren. Da die Sieben-Segment-Anzeigen sehr schnell hintereinander ein- und ausgeschaltet werden, nimmt das leicht zu trügende Auge ständig erleuchtete Ziffern wahr.

● **Das Anzeigefeld:** Zur Zeit sind LCDs (Flüssigkristallanzeigen) sehr aktuell, speziell für batteriebetriebene Geräte. Wir haben uns trotzdem für Leuchtdioden entschieden – sie haben einige Vorteile gegenüber ihren Konkurrenten: LEDs sind leichter anzuschließen, besonders was die Steuerung des Dezimalpunktes angeht. Flüssigkristall-Anzeigen benötigen außerdem ein Wechselstrom-Signal und zusätzliche Logik-Schaltungen für die Positionierung des Dezimalpunkts.

A/D-Wandlung und Steuerung des Displays



Feine Zutaten

Bevor wir mit dem Zusammenbau unseres Digitalvoltmeters beginnen, wollen wir den 7135 A/D-Wandler-Chip etwas genauer studieren: Wir erläutern die Pinbelegung und die Fähigkeiten des Chips.

Wir beziehen uns zunächst auf den Schaltplan von Seite 2027, der die Pinbelegung und die Bezeichnung der Anschlüsse darstellt. Die Erklärungen erfolgen in der Reihenfolge der Pin-Nummern.

1. **V−:** Der 7135 braucht eine positive und eine negative Versorgungsspannung. Die positive Versorgungsspannung darf +6 Volt nicht überschreiten, auf der negativen Seite ist −9 Volt das Maximum. Zur Vereinfachung werden wir mit +5/−5V arbeiten. Die Stromaufnahme für die negative Versorgung beträgt nur circa 0,8 mA. Dadurch kann die Schaltung der Spannungsquelle sehr einfach aufgebaut sein. In unserem Beispiel verwenden wir ein Netzteil, es geht aber auch mit einer 6-Volt-Batterieversorgung (sowohl TTL- als auch CMOS-ICs arbeiten bei sechs Volt ohne Probleme).

2. **Reference:** Die Referenzspannung für den 7135 muß sehr stabil sein. Der Meßbereich eines Dual-Slope-Wandlers ist genau doppelt so groß wie der Wert der Referenzspannung – für einen Meßbereich von 2V brauchen wir also eine Referenzspannung von genau 1,000 Volt. Woher also nehmen?

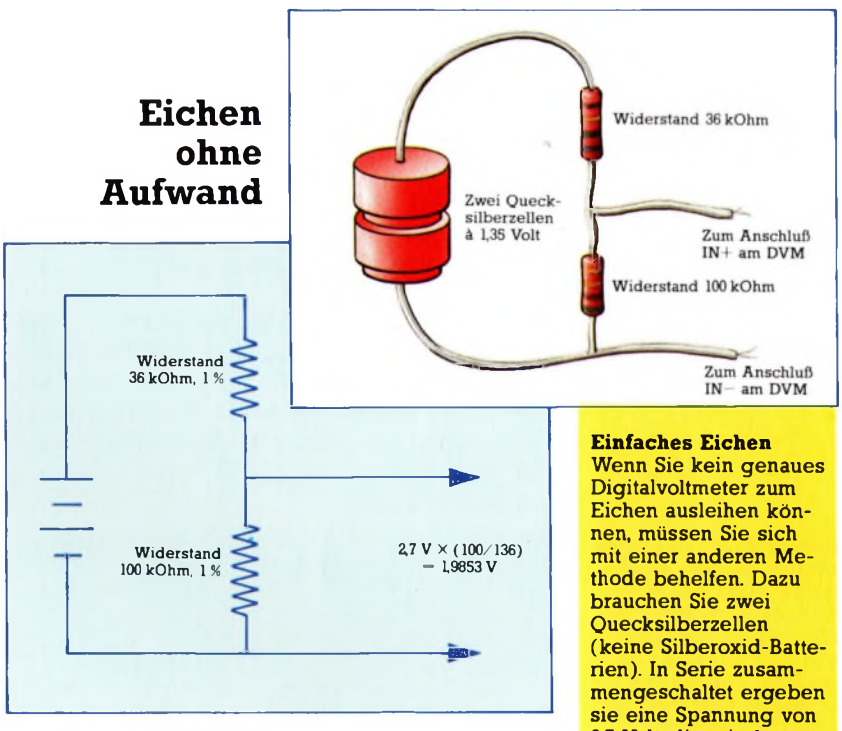
Präzision gefordert

Um eine 1-Volt-Referenzspannung zu erzeugen, gibt es mehrere Verfahren. Wir empfehlen die Verwendung einer Präzisions-Zenerdiode 1,22 Volt (Typ 9491). Die erforderlichen 1 Volt werden mit einem Potentiometer abgegriffen. Als Potentiometer sollte aber keinesfalls ein billiger Lautstärkeregler verwendet werden, sondern ein 10- oder mehrgängiges Trimmerpoti von möglichst hoher Qualität. Leihen Sie sich ein gutes Digitalvoltmeter (DVM), und stellen Sie damit am Trimmer die 1 Volt genau ein. Noch besser wäre es, am DVM mit einem Spannungsteiler die Anzeige auf 1,9999 Volt zu bringen und diese Spannung dann an das selbstgebaute Voltmeter anzuschließen: Danach wird auch hier der Anzeigewert ganz exakt auf 1,9999 Volt eingestellt.

Wenn Sie keinen Zugang zu einem genauen Digitalvoltmeter haben, können Sie die Eichspannung auch aus zwei in Serie geschalteten Quecksilberzellen mit einem Spannungsteiler erzeugen. Den Spannungsteiler bauen Sie mit einem 36 kOhm- und einem 100 kOhm-Widerstand auf. Sie müssen dazu möglichst eng tolerierte Widerstände (ein Prozent oder weniger) und selbstverständlich nagelneue Quecksilberzellen verwenden.

3. **Analogue Ground:** Dieser unschuldig wirkende Pin kann mehr Schwierigkeiten hervorrufen, als es auf den ersten Blick scheint. Das liegt daran, daß es zwei unterschiedliche Masseleitungen gibt: die Analog-Masse und die Digital-Masse (Pin 24). Die analogen Signale, + und − Versorgungsspannung und die Digital-Leitungen müssen strikt voneinander getrennt sein. Wir brauchen dazu einen Schaltungsaufbau, der verhindert, daß digitale Signale in den Analogteil und umgekehrt gelangen, wobei der zweite Fall weniger gravierende Folgen hat.

Alle Punkte des Analogteils, die auf Massepotential liegen (die Verbindung zwischen der Referenzdiode und dem Poti-Anschluß mit Pin 3), und die Masseseite des an der Diode angeschlossenen Widerstandes (der mit dem Integrationskondensator und dem Auto-Nullstellungskondensator an Pin 4 und 5 verbunden ist) sollten an einem einzigen Lötpunkt zusammentreffen. Auch die Anode der Referenzdiode und den 100 KOhm-Widerstand (der mit dem Pin Int.Out über eine Diode verbunden ist) sollten Sie so zusammenfassen.



Einfaches Eichen
Wenn Sie kein genaues Digitalvoltmeter zum Eichen ausleihen können, müssen Sie sich mit einer anderen Methode behelfen. Dazu brauchen Sie zwei Quecksilberzellen (keine Silberoxid-Batterien). In Serie zusammengeschaltet ergeben sie eine Spannung von 2,7 Volt, die mit dem Spannungsteiler aus den beiden Widerständen auf 1,9853 Volt reduziert wird. Die Abweichung der Spannung liegt bei dieser Anordnung im Bereich von 1–2 Prozent, sofern Metallschicht-Widerstände verwendet werden. Sie sollten aber nicht versuchen, den 36 kOhm-Widerstand aus mehreren Einzelbauteilen zusammenzulöten – Sie würden die Meßgenauigkeit dabei verschlechtern. Mit dem Trimmerpoti stellen Sie nach dem Anschluß der Eichspannungsquelle die Anzeige exakt auf 1,9999 Volt.

Genauso wichtig ist es, die digitalen Massepunkte auf einer Lötstelle zu vereinigen. Im Schaltbild haben wir zwei verschiedene Symbole für die beiden Massepunkte verwendet: für die analoge und für die digitale Masse. Beide Massepunkte werden mit einem möglichst dicken Kupferdraht verbunden.

Wenn Sie hier Fehler machen, treten vielfältige Komplikationen auf: Die Anzeige im Display wird ungenau, das Display könnte mehrere verschiedene (falsche) Anzeigen erzeugen, das Voltmeter könnte das Messen aber auch ganz verweigern, ins Schwingen geraten oder gar beschädigt werden.

Die Stromversorgung des Taktgenerators, des Display-Treibers und des Displays selbst sollte durch den Einsatz von Elektrolyt-Kondensatoren von hoher Kapazität entkoppelt und durch zusätzliche keramische Scheibenkondensatoren gegen hochfrequente Störungen geschützt werden. Genauso wichtig ist es, daß die analogen Schaltungsteile auf der Platine möglichst weit von den digitalen Signalleitungen (etwa CLOCK, STROBE und BUSY) entfernt sind.

Die Regel lautet, digitale und analoge Schaltungsteile physikalisch zu trennen und auf die Führung der Masseleitungen zu achten – Erdschleifen müssen um jeden Preis vermieden werden.

Weiter gilt, daß man möglichst keine unbenutzte digitale Signal-Leitung (STROBE, UNDERRANGE, RUN/HOLD u. ä.) undefiniert lassen sollte. Besser ist es, die freigebliebenen Anschlüsse mit einem 4,7 KOhm-Widerstand auf High oder mit einem 440 Ohm-Widerstand auf Low zu halten.

4. Int. Out. (Integrator-Ausgang): Integrierende A/D-Wandler wie der 7135 beruhen auf der Annahme, daß der Spannungsanstieg über dem integrierenden Kondensator dem Stromfluß proportional ist. Der absolute Wert dieses Kondensators ist nicht so wichtig – zwischen 0,4 und 0,47 µF (Mikrofarad) geht alles bestens. Worauf es ankommt, sind möglichst geringe Verluste im Kondensator. Am besten eignen sich daher Polypropylen-Kondensatoren, es geht aber auch mit Polystyren-Kondensatoren oder Polycarbonat-Typen. Nur Elektrolyt-Kondensatoren funktionieren nicht.

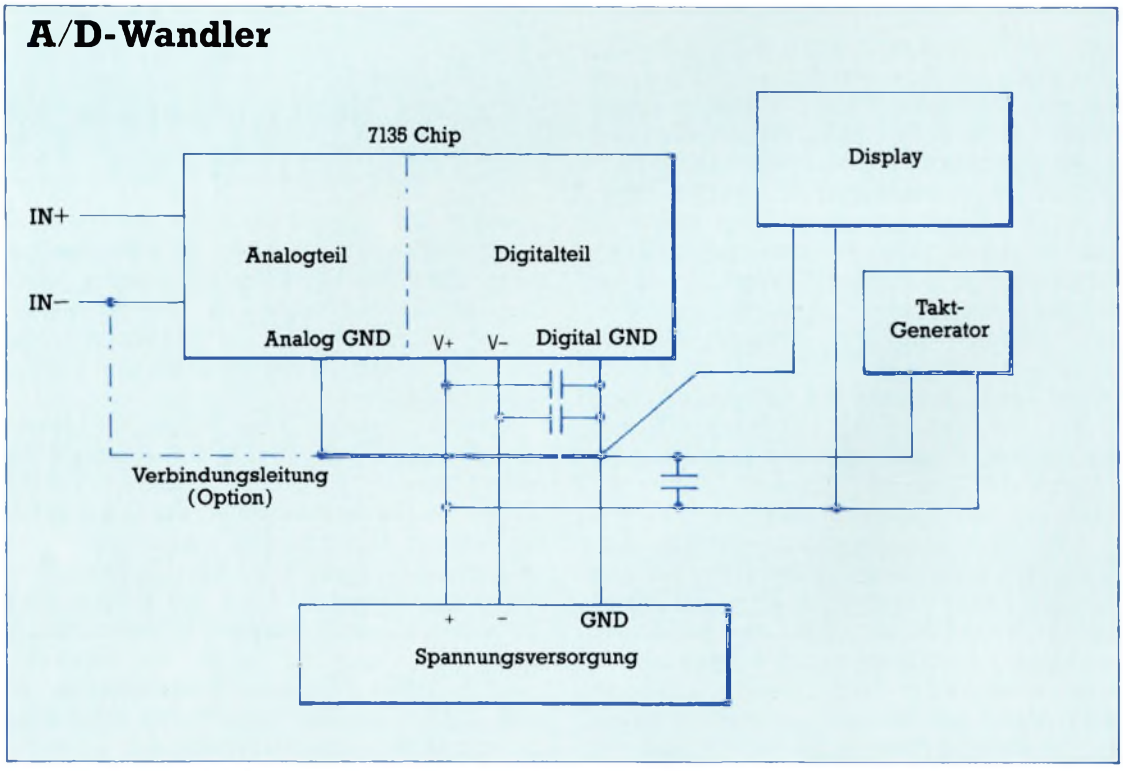
5. A–Z IN: Der hier verwendete Kondensator darf von jeder Art und Güte sein. Nicht einmal sein Wert ist kritisch – je größer, desto besser. Aber auch hier keinen Elko einsetzen!

6. BUF OUT (Bufferausgang): An diesem Punkt wird der Integrator-Widerstand angeschlossen. Den Wert des Widerstandes ergibt:

$$R = \frac{\text{Spannung für Vollausschlag}}{20\ \mu A}$$

Dafür kann jeder 100 KOhm-Widerstand verwendet werden. Ein Metallschichtwiderstand mit einer Abweichung unter einem Prozent eignet sich aber besser als ein billiger Kohleschicht-Widerstand.

Die Diode am Integrator-Ausgang dient zur Korrektur kleiner roll-over-Fehler. Hier kann jedes Bauteil zum Einsatz kommen. Bastler mit geringen Erfahrungen wünschen sich zwar meist eine Typenbezeichnung, es geht aber wirklich mit jeder verfügbaren Diode – mit Siliziumdioden allerdings besser als mit Germanium-Typen und mit Signaldioden besser als mit Gleichrichtern. Geeignete Dioden kann man tütenweise für wenig Geld bekommen.



Beim Aufbau der Schaltung muß sehr genau darauf geachtet werden, daß der Analogteil des Voltmeters sicher von den digitalen Elementen des Gerätes getrennt ist. Die einzelnen Segmente dürfen keinen Kontakt zueinander haben. Das abgebildete Schaltungsschema minimiert die Möglichkeit von Masseschleifen und einer eventuellen Rückkopplung von Digitalsignalen in den Analogteil.

Signalabtastung

Wir beschließen unsere detaillierte Untersuchung der einzelnen Anschlüsse des 7135-Wandlers und bauen eine Zusatzschaltung für einen zweiten Taktgenerator-Kreis auf, mit der sich Messungen schneller durchführen lassen.

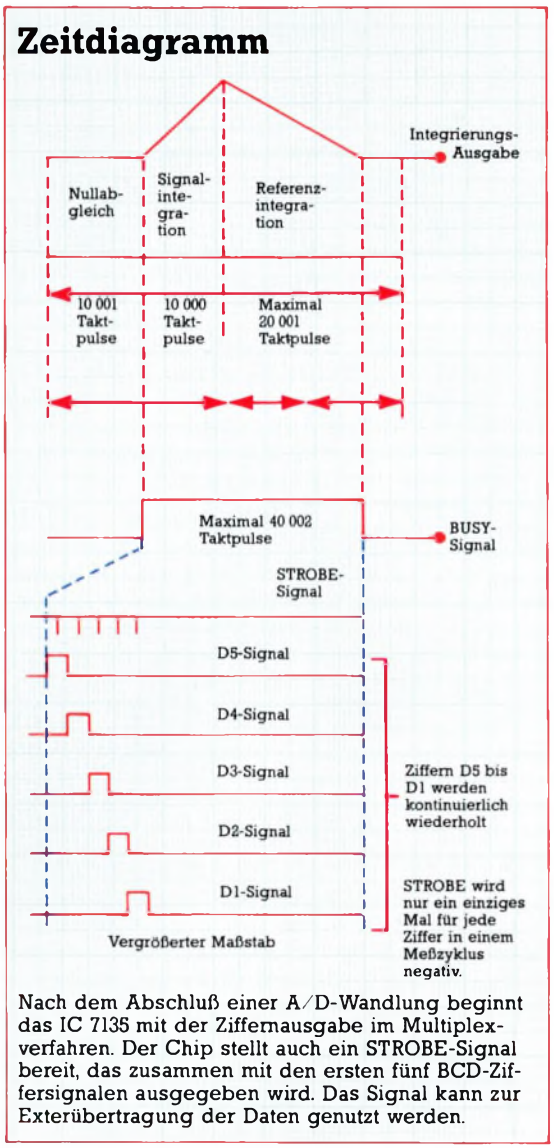
Die Bauteile im einzelnen: 7/8. REF.CAP: Die Kapazität und auch die Qualität dieses Bauteils ist nicht sehr kritisch. Wir schlagen einen Kondensator von 1µF vor. Sie können aber auch eine geringere Kapazität wählen – das hat nur zur Folge, daß Sie nach einer Überschreitung des Meßbereiches einige Sekunden länger warten müssen, bis sich das Gerät wieder „erholt".

9/10. IN−/IN+: Diese beiden Anschlüsse gehören zum analogen Masseanschluß (Pin 3). Es können Spannungen zwischen −4 Volt und +4,5 Volt verarbeitet werden. Diese Spannungen können erdfrei sein, oder der IN-Pin kann mit der (analogen) Masse verbunden werden.

11. V+: Zur Versorgung des ICs ist hier der Anschluß einer 5-Volt-Spannungsquelle erforderlich, die allerdings nur 3mA liefern muß.

12. D5: Es gibt fünf digitale enable-Ausgänge (D1–D5). (Die weiteren Ausgänge liegen auf den Pins 17–20.) Sie werden dafür genutzt, die einzelnen Stellen des LED-Displays nacheinander zu aktivieren, was für den Multiplex-Betrieb erforderlich ist. Jedes Ziffern-enable wird nacheinander für 200 Taktpulse High. Dabei wird von D5, dem höchstwertigen Bit (MSB), zu D1 (niedrigstes Bit) gezählt. Wenn einer der Anzeigentreiber-Ausgänge aktiv wird, steigt seine Spannung von etwa 0,25 Volt auf rund 4 Volt an. Der angeschlossene Transistor wird durchgesteuert, und der Strom gelangt zum Anzeigeelement der entsprechenden Dezimalstelle. Die Display-Treiberausgänge aktivieren die Segmente nur, die Ansteuerung erfolgt mit BCD-Signalen (B1, B2, B4 und B8), die von dem Siebensegment-Decoder/Treiber 7447 geliefert werden. Mit logischen Schaltungen, die ein Vierleitungs-BCD-Signal zum Ansteuern einer Sieben-Segment-Anzeige umwandeln, haben wir uns bereits befaßt.

13. B1 (LSB): Es gibt vier BCD(Binär codierter Dezimalwert)-Ausgangsleitungen: B1 auf Pin 13, B2 auf Pin 14, B4 auf Pin 15 und B8 auf Pin 16. Gemeinsam liefern sie die darzustellende, binär codierte Dezimalzahl. Die Ziffern-enable-Signale sind so synchronisiert, daß immer nur die gerade aktive Ziffer mit dem für sie richtigen Anzeigewert belegt wird.



Nach dem Abschluß einer A/D-Wandlung beginnt das IC 7135 mit der Ziffernausgabe im Multiplexverfahren. Der Chip stellt auch ein STROBE-Signal bereit, das zusammen mit den ersten fünf BCD-Ziffersignalen ausgegeben wird. Das Signal kann zur Exterübertragung der Daten genutzt werden.

Alle fünf Ziffern scheinen auf dem Display gleichzeitig dargestellt zu werden, in Wahrheit sind sie jedoch im Multiplex-Verfahren angesteuert und erscheinen in schneller Folge nacheinander. Dadurch leuchten die LED-Ziffern schwächer als bei dauerhafter, gleichmäßiger Ansteuerung. Das ist jedoch kein Fehler: Der Stromverbrauch der LEDs sinkt auf ein Fünftel ab und hilft Batterien sparen.

Für die gleichzeitige Steuerung aller Ziffern wären normalerweise 20 BCD-Ausgangsleitungen erforderlich. Damit bräuchten wir auch fünf 7447-Treiber ICs. Das Multiplex-Verfahren erspart also nicht nur Leitungen, sondern auch eine Vielzahl von Bauteilen.

21. BUSY: Diese Leitung ist besonders dann wichtig, wenn das Digitalvoltmeter mit dem

Computer in Verbindung treten soll. Wenn die Signalleitung BUSY auf High-Potential liegt, bedeutet das, daß der A/D-Wandler gerade mit Berechnungen beschäftigt ist, also die Daten auf Ziffern- und BCD-Ausgängen momentan keine definierte Bedeutung haben. Zu Beginn jeder Signal-Integrationsphase wird der BUSY-Ausgang High und bleibt bis zum ersten Taktimpuls nach dem Nulldurchgang in diesem Zustand. Erst wenn er wieder auf Low liegt, befinden sich die Signalleitungen in einem stabilen Zustand mit gültigen Werten, die vom Computer gelesen werden können.

22. **CLOCK IN:** Zur Steuerung des 7135 ist ein externer Taktgenerator erforderlich. Die Taktgeschwindigkeit sollte 1,2 MHz nicht überschreiten – das ist allerdings schon recht schnell, die gemessene Spannung wird also nach sehr kurzer Zeit von neuem überprüft. Eine so hohe Frequenz erzeugt allerdings zusätzliche Probleme bei der Auswahl von Bauteilen. Der von uns verwendete Chip 555 kann die Grenzen der Geschwindigkeit des Wandlers nicht voll ausnutzen: Er erzeugt nur Frequenzen bis maximal 100 kHz.

Da ein vollständiger Meßzyklus maximal 40 002 Taktpulse dauert, bestimmt die Taktfrequenz die Geschwindigkeit der Messung. Bei einer Taktfrequenz von 100 kHz wären das etwa 2,5 Messungen pro Sekunde. Wenn Sie schneller messen möchten, können Sie die Frequenz auf 1,2 MHz steigern, dazu brauchen Sie allerdings einen zweiten Schaltkreis zur Erzeugung der Taktsignale. Die Kondensatoren für automatische Nullstellung und Referenz müssen zudem von extrem guter Qualität sein. Die Meßgenauigkeit wird bei hoher Geschwindigkeit schlechter. Der 7135-Chip arbeitet mit Taktfrequenzen bis herunter zu 1 Hz – dann müßten Sie allerdings 40 002 Sekunden auf Ihr Ergebnis warten!

23. **POLARITY:** Dieser Pin gibt ein Signal aus, das die Polarität der gemessenen Spannung angibt. Bei allen positiven Meßwerten einschließlich der 0,0000 ist das Signal positiv. Es wird genutzt, um das Plus-Zeichen in der linken, ½stelligen Ziffernanzeige des Displays zu aktivieren.

24. **DIGITAL GROUND:** (Digitale Masse): Siehe Pin 3 (Analoge Masse=ANALOGUE GROUND)

25. **RUN/HOLD:** Liegt dieser Eingang auf High, dann befindet sich der Wandler-Chip im Freilaufbetrieb und gibt seine Meßwerte nach jeweils 40 002 Taktpulsen aus. Wird er auf LOW geschaltet (entweder durch ein HOLD-Signal vom Rechner oder durch einen Schalter am Voltmeter), wird die aktuelle Anzeige festgehalten. Nach dem Anhalten startet ein mindestens 300 Nanosekunden andauernder positiver Wert an diesem Pin einen neuen Meßzyklus. Für den Anschluß über langsame Parallelschnittstellen ist dieser Pin ideal, wenn nämlich nicht alle fünf BCD-Ausgangssignale gleichzeitig verarbeitet werden können.

Eine Vereinfachung

26. **STROBE:** In jedem Meßzyklus geht das STROBE-Signal fünfmal auf Minus, für jede gültige Ziffer des Meßwertes einmal. Die STROBE-Pulse erscheinen in der Mitte jedes Ziffern-enable-Ausgangssignals (siehe Ablaufplan). Das STROBE-Signal soll die Verbindung zu externen Microprozessoren vereinfachen und kann genutzt werden, um die BCD-Daten in Zwischenspeicher zu schaffen. STROBE wird nur für einen halben Taktzyklus lang negativ (aktiv) und zwar für jede Ziffer nur einmal. Die Displaytreiber und BCD-Ausgänge wiederholen ihre Ausgabe während eines Meßzyklus mehrfach (um eine dauerhafte Anzeige des Wertes zu ermöglichen).

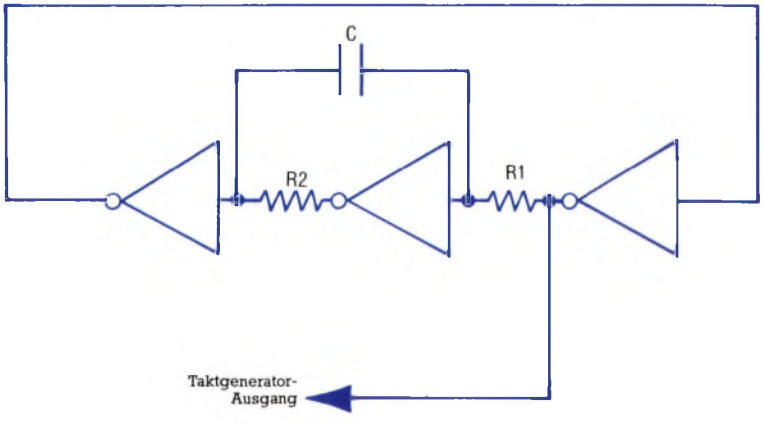
27. **OVER RANGE** (Meßbereichsüberschreitung): Dieses Signal zeigt eine Überschreitung des zulässigen Meßbereiches (mehr als 20 000 Impulse) an. In komplizierteren Multimetern könnte man es für eine automatische Bereichswahl einsetzen – eine Schaltung, die selbsttätig ohne mechanische Schalter die Eingangsempfindlichkeit des Instruments verändert. In unserer einfachen Anwendung soll es nur zur Steuerung einer Leuchtdiode eingesetzt werden, die uns warnt, wenn die Abschwächerschaltung am Eingang des Multimeters auf einen niedrigeren Wert gestellt werden muß.

28. **UNDER RANGE** (Meßbereichsunterschreitung): Dieses Signal geht auf HIGH, wenn die Eingangsspannung zu niedrig ist. Wie das OVER RANGE-Signal ist es zuerst einmal dafür gedacht, eine automatische Bereichsumschaltung zu steuern. Voltmeter mit automatischer Bereichswahl nutzen diese beiden Signale für die Selektion des Meßbereichs. Für uns wäre das etwas zu schwierig. Wir verzichten daher auf diese Funktion.

Die Grundlagen der Schaltung des Digitalvoltmeters sind damit recht ausführlich dargestellt worden. Im nächsten Abschnitt geht es an den konkreten Aufbau des Gerätes.

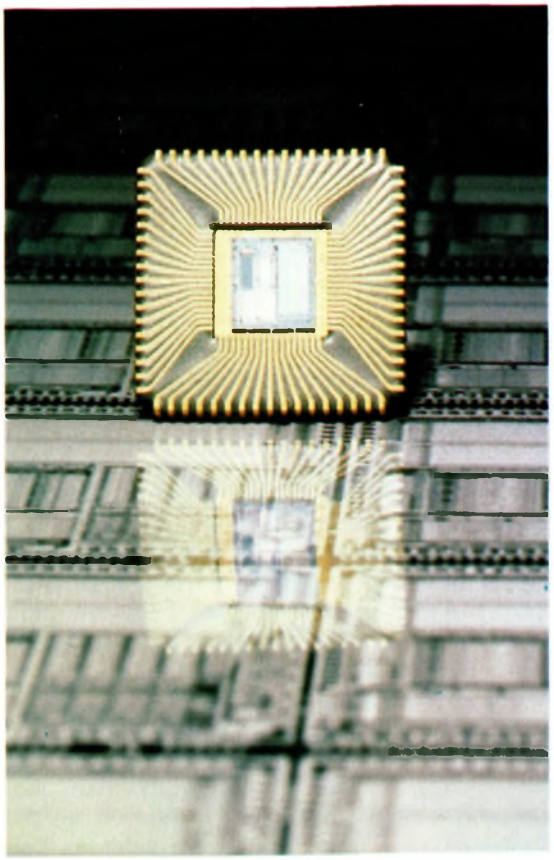
Diese Schaltung kann alternativ zu dem von uns vorgeschlagenen Timer-Chip 555 eingesetzt werden. Sie besteht aus drei TTL-Invertierern, zwei Widerständen und einem Kondensator. Die Formel zur Berechnung der gewünschten Frequenz lautet:
F = 2 C (R1 + R2)
Da der 7135-Wandler mit recht hoher Geschwindigkeit arbeitet, können Sie diesen Taktgeber einsetzen, wenn es mehr auf hohe Meßraten als auf exakte Ergebnisse ankommt.

Taktgenerator für höhere Frequenzen

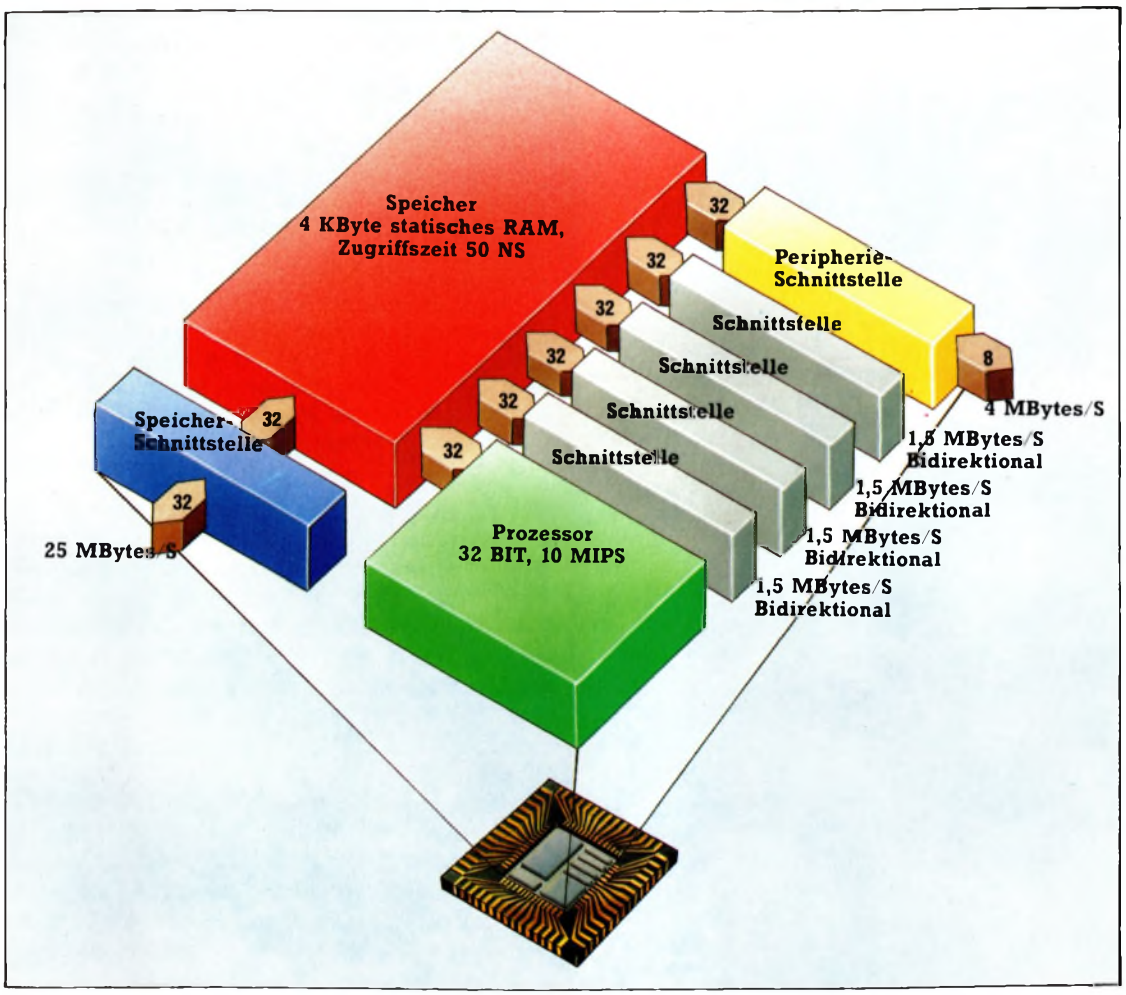


Schnelle Schritte

Der Einfluß, den die Entwicklung von Transputern auf die Computertechnik hat, läßt sich nur mit der Entdeckung des Transistors in den fünfziger Jahren vergleichen. Sie erlauben höchste Verarbeitungsgeschwindigkeiten.



Die Erfindung des Transistors zu Anfang der fünfziger Jahre bereitete den Weg zur Serienproduktion von Computern. Ein Microprozessor besteht heute aus zahlreichen identischen Transistoren, die auf einem einzelnen Chip untergebracht sind. Der Inmos Transputer ist ein Prozessor, der ähnlich wie ein Transistor angewandt werden soll – als Komponente oder Grundbaustein für größere Systeme. Der Name ergibt sich aus TRANSistor und ComPUTER. Eine Verbindung mehrerer



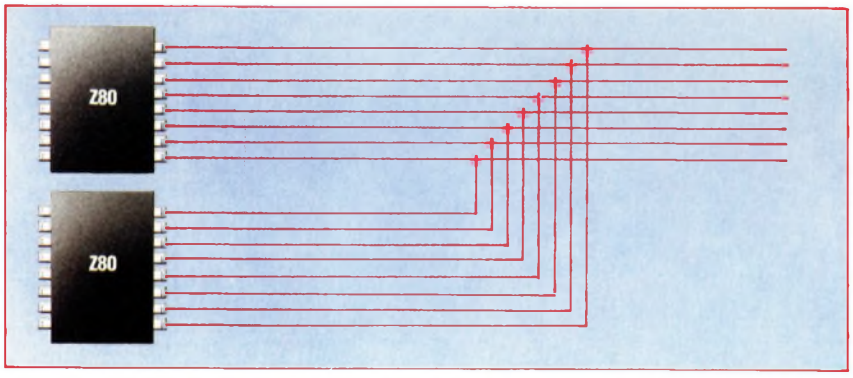
Von der Firma Inmos stammt ein neuer Stern am IC-Himmel: Der Transputer ist ein kompletter 32-Bit-Computer auf einem einzigen Chip. Neben dem RAM-Speicher sind auch vier serielle Schnittstellen integriert. Transputer sind für den Einsatz in größeren Parallelrechnern vorgesehen. Durch die seriellen Schnittstellen lassen sich viele Transputer zu einem Netzwerk zusammenfassen. Der Aufbau eines solchen Netzwerkes kann die unterschiedlichsten Strukturen aufweisen – allein die Problemstellung determiniert die Rechnerarchitektur.

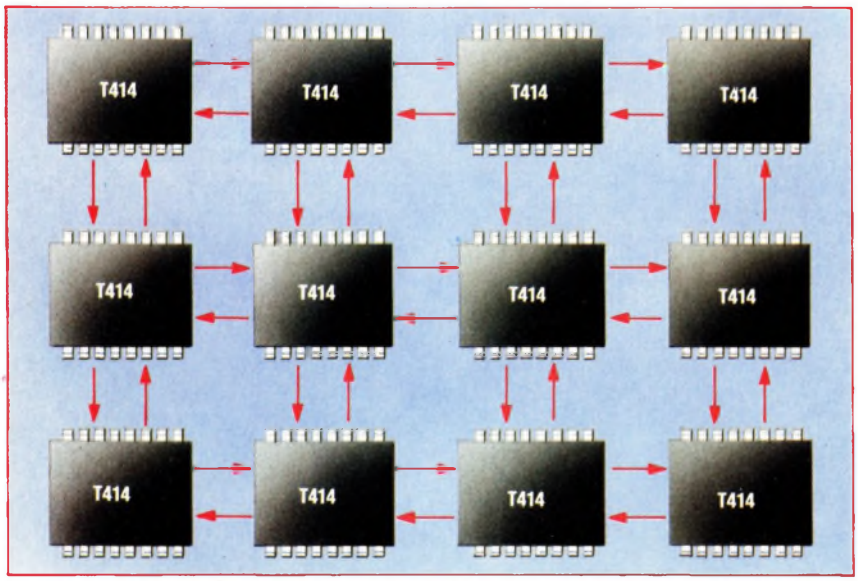
Transputer macht Parallelverarbeitung von Daten möglich, wobei jeder Transputer eine getrennt programmierbare Einheit darstellt.

Die Zusammenschaltung funktioniert, weil Transputer alle Komponenten eines vollständigen Computers vereinigen. Jeder Transputer hat eine CPU, Speicherplatz und serielle Datenschnittstellen. Dadurch kann er Programme selbstständig ausführen, wobei als externe Unterstützung nicht mehr als Strom und Clock-Pulse nötig sind. Transputer können sich zudem über ihre seriellen Datenverbindungen einfacher verständigen als die üblichen CPUs.

Ein herkömmlicher Microprozessor wie etwa der Z80 „beherrscht" jede Schaltung, in der er eingesetzt wird. Die Verständigung mit der Außenwelt erfolgt über den prozessorgesteuerten Parallelbus. Zwei Z80 zu einer Unterhaltung zu bewegen, ist aus zwei Gründen schwierig: Zunächst gibt es Platzprobleme auf der Karte, wenn viele Prozessoren denselben Bus benutzen sollen – jeder Anschluß braucht eben acht Leitungen. Bei den modernen 16- oder 32-Bit-CPUs wird dieses Problem natürlich noch entsprechend größer.

Eine weitere Schwierigkeit ist, daß jeder Z80 den Bus allein kontrollieren möchte. Es muß also durch Software verhindert werden, daß beide Prozessoren gleichzeitig senden. Außerdem ist ein Z80 beim Datenaustausch über den Bus für weitere Aufgaben nicht verfügbar.





Von Neumann-Architektur

Alle modernen Computer – bescheidene Homecomputer genau wie Super-Rechner – funktionieren nach den Neumannschen Prinzipien. Ein Computer nach dem Neumann-Prinzip verfügt über einen zentralen Prozessor (CPU), der mit einem Speicher verbunden ist. Im Speicher stehen sowohl die auszuführenden Befehle als auch Daten, die mit diesen Befehlen verarbeitet werden sollen. Die Befehle werden nacheinander (sequentiell) abgerufen und von der CPU ausgeführt.

Dieser Aufbau, der von Neumanns Namen trägt, ermöglichte die Konstruktion von Multifunktionsanlagen. Wichtig ist dabei, daß die Zahlen im Speicher sowohl Programme als auch Daten sein können. Dadurch bekamen Computerkonstrukteure einen zusätzlichen Freiraum. Bisher mußten sie nämlich ihre Apparate für jede Spezialaufgabe wie etwa Entschlüsselung oder Maschinensteuerung neu konstruieren.

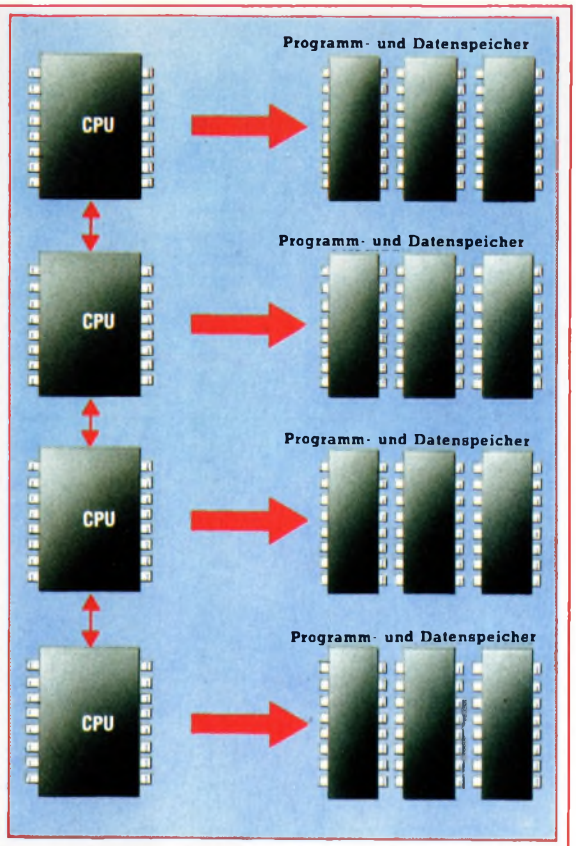
Das Problem bei einem von-Neumann-Rechner liegt in der sequentiellen Befehlsverarbeitung. Wie schnell man die CPU auch macht, sie muß ihre Befehle und Daten aus dem Speicher holen. Hierbei gibt es physikalische Grenzen – die Geschwindigkeit, mit der ein einzelner Microprozessor Daten mit dem Speicher austauschen kann, wurde zum Engpaß.

Dieser Engpaß kann durch eine Vielzahl von gleichzeitig arbeitenden CPUs umgangen werden. Die Technik dahinter wird als „Parallelverarbeitung" bezeichnet. Trotz beschränkter Lesegeschwindigkeit können 100 CPUs mit 100 Speichereinheiten eben auch hundertmal schneller arbeiten als ein einzelner Prozessor. Parallele Rechner werden zusätzlich dadurch beschleunigt, daß jede einzelne CPU weniger mit dem Speicher kommunizieren muß.

Parallelverarbeitung hat natürlich auch ihre spezifischen Schwierigkeiten. Um ein Programm auf mehreren CPUs laufen zu lassen, muß jeder Prozessor wissen, was er zu tun hat. Er muß seinen Teil abarbeiten, ohne mit der Tätigkeit seiner Kollegen zu kollidieren

Beim Transputer ist dieser Vorgang einfacher – er braucht nur zwei Leitungen, um mit einem „Kollegen" in Kontakt zu treten. Jeder Transputer hat zudem vier serielle Schnittstellen, er kann also mit vier anderen Chips gleichzeitig kommunizieren. Damit sind zwei- oder gar dreidimensionale Felder von Transputern bei minimalem Verdrahtungsaufwand realisierbar.

Transputer brauchen bei ihrer Kommunikation keine spezielle Software, um Prioritätsfragen zu klären. Ein einzelner Transputer kann vieles gleichzeitig erledigen – etwa Daten von allen vier Schnittstellen empfangen, ohne seine anderen Aufgaben deshalb nur einen Sekundenbruchteil zu unterbrechen.



oder sie zu überholen.

Wenn wir zum Beispiel den Ausdruck

(3+4)*(9-7+3)

von zwei Prozessoren auswerten lassen wollen, könnte jeder einen Klammerausdruck berechnen, und einer von beiden die abschließende Multiplikation vornehmen.

Natürlich muß vorher festgelegt sein, welche CPU den letzten Schritt ausführt. Wenn das CPU1 ist, darf sie ihre letzte Berechnung nicht ausführen, bevor CPU2 fertig ist. Die CPUs müssen sich also verständigen, also auch Teilergebnisse austauschen können. Solche Probleme lassen sich mit herkömmlichen Programmiersprachen nur ausgesprochen mühsam lösen.

Einsame Spitze

Technologie: Der T414 wird in 1.5 Mikron CMOS-Technik hergestellt. Dadurch ist die Packungsdichte größer als bisher üblich.
Prozessor: 32-Bit-Microprozessor mit etwa zehn Millionen Befehlen pro Sekunde (etwa 10mal schneller als etwa der Motorola 68000). Arbeitet mit 70 Befehlen, die OCCAM direkt unterstützen.
Speicherplatz auf dem Chip: Statisches RAM mit 50 ns Zugriffszeit. Wird benutzt, um kürzere OCCAM-Prozesse durchzuführen; ersetzt die Register konventioneller Microprozessoren. Adressierbarer Speicher: bis zu vier Gigabytes. Damit kann ein Transputer als Einzelrechner etwa in PCs eingesetzt werden. Der Prozessor unterscheidet nicht zwischen internem und externem Speicher. Einzige Ausnahme ist die Zugriffszeit.
Serielles Interface: Jedes Interface kann bidirektional zehn Megabits pro Sekunde verarbeiten. Damit haben die vier Schnittstellen eine Kapazität von 80 Megabit pro Sekunde. Der Transfer verläuft asynchron mit Hardware-Handshake; Clocksignale müssen daher nicht berücksichtigt werden.
Gleichzeitigkeit: Ein einzelner T414 kann mehrere Aufgaben gleichzeitig wahrnehmen. Er wird dabei von einem Prozeßsteuer-Modul überwacht. Alle Bereiche des Chips arbeiten unabhängig voneinander. Der Prozessor kann also auch dann Rechnungen durchführen, wenn Daten über das Interface ein- oder ausgehen.
Weiterentwicklung: Der T414 ist der erste aus einer Familie kompatibler Transputer, bei denen einzelne Speichergruppen oder Schnittstellen durch spezielle Funktionsgruppen ersetzt werden sollen. Der Grafik-Prozessor G213 und das Controller-IC M212 stehen kurz vor der Fertigstellung.
Die Firma: Inmos wurde 1978 mit Unterstützung der englischen Regierung gegründet und gehört heute zur Thorn EMI Gruppe. Entwurf und Entwicklung der Transputer und der Programmiersprache OCCAM wurde größtenteils in Bristol durchgeführt, produziert wird in Newport, Wales.

Diese Arbeitsweise erlaubt es, große Programme in Einzelteile zu zerlegen, die dann auf mehreren Transputern simultan abgearbeitet werden. Jeder Transputer kann seinen Teil des Programms im eigenen Speicher ablegen und die „Kollegen" zum gegebenen Zeitpunkt über ein Interface informieren. Muß ein spezieller Programmteil besonders schnell ausgeführt werden, ist nur ein vergrößerter Einsatz von Transputer-Arbeitskräften notwendig.

Das Geheimnis, das Transputer zu all diesem befähigt, liegt in der Sprache, in der sie programmiert werden. Anders als ein traditioneller Microprozessor wird ein Transputer nicht in Assembler, sondern in einer speziell entwickelten Sprache namens OCCAM programmiert. Sie ist auf Parallelverarbeitung zugeschnitten. Ein OCCAM-Programm setzt sich aus einzelnen „Prozessen" zusammen, die etwa einer Subroutine oder Prozedur in BASIC oder PASCAL entsprechen. Im Unterschied zu BASIC oder PASCAL können mehrere Prozesse gleichzeitig ausgeführt werden.

OCCAM kann Daten nicht nur über Variablen, sondern auch über „Kanäle" austauschen. Damit stehen Ergebnisse aus verschiedenen, gleichzeitig ablaufenden Prozessen allen angeschlossenen Transputern sofort zur Verfügung. Die Kanäle sind so intelligent aufgebaut, daß keine Information ausgetauscht werden kann, ohne daß beide Partner dafür bereit sind. Dabei spielt es für OCCAM keine Rolle, ob ein Prozeß auf demselben Transputer abläuft oder auf einem anderen.

Wer im Computerbereich eine Stellung sucht und keine Spezialausbildung mitbringt, hat beim kleineren Hardware-Einzelhändler noch die besten Aussichten auf einen Job. Das Problem einer solchen Beschäftigung liegt oft in der Überlastung mit täglichem Kleinkram; der Angestellte hat kaum Zeit für Fortbildung und Information über neue Produkte.

Im Vertrieb

Der komplizierte und unübersichtliche Computermarkt hat durch ständiges Wachstum eine Vielzahl von Stellen im Marketing- und Vertriebsbereich geschaffen. Beide Berufsfelder bieten interessante Aufstiegsmöglichkeiten und hohe Gehälter.

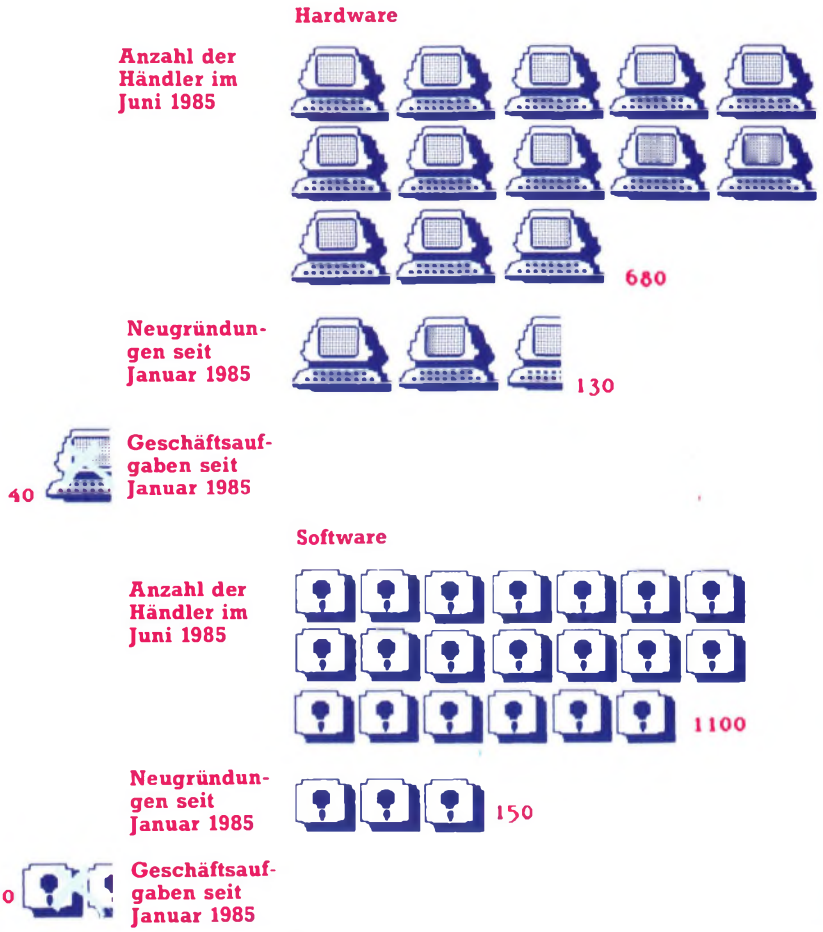
Die ungewöhnlichen Wachstumsraten im Computergeschäft beginnen sich langsam zu normalisieren, doch schrumpft damit auch der Zuwachs an offenen Stellen. Zwar werden immer noch dreimal soviele Unternehmen neu gegründet, wie sich (zuletzt und spektakulär Sinclair) aus dem Geschäft zurückziehen. Aber auch der beste Programmierer oder Vertriebsmann ist heute vor einer überraschenden Firmenschließung nicht sicher.

Arbeitsstellen im Vertriebs- oder Marketingbereich gehören zu den bestbezahlten Positionen dieser Branche. An der Spitze steht dabei der Vertriebsprofi, der mit kommerziellen Firmen über die Anschaffung größerer Anlagen oder umfangreicher Software-Systeme verhandelt. Er kann im Jahr um hunderttausend Mark verdienen – zumeist als Provision. Marketing-Angestellte, die keine Provision erhalten, verdienen weniger. Erst nach mehrjähriger Tätigkeit kann man den Sessel eines Marketing-Direktors erkämpfen, bei dem das Salär üblicherweise die 100 000-Markgrenze überschreitet. Vertrieb und Marketing sind häufig der Einstieg karrierebewußter Aufsteiger gewesen, die sich von hier bis an die Spitze bedeutender Unternehmen vorarbeiten konnten. Bis vor kurzem stammten beispielsweise noch sämtliche IBM-Manager aus dem Vertriebsbereich des Unternehmen.

In den meisten Laufbahnen ist es die schwierigste Aufgabe, die erste Stufe der Karriereleiter zu erklimmen; der Vertrieb macht hier keine Ausnahme. Üblicherweise erwarten die Firmen eine gewisse Berufserfahrung. Wie aber soll der Neuling sie erwerben, wenn ihm nirgends eine Chance geboten wird?

Personalberatungsfirmen empfehlen den Bewerbern um Vertriebsjobs meist den Besuch von Computer-Einführungskursen, die teilweise auch von den Arbeitsämtern gefördert werden. Der nächste Schritt ist eine Anstellung bei einem Unternehmen, das etwa Büroausstattung oder Computer verkauft. Hier wird sel-

Zuwächse beim Handel



Trotz eher pessimistischer Vorhersagen mancher Fachleute hat sich die Wachstumsrate bei den Computer-Händlern bis heute nicht verlangsamt. Es ist eher das stabile Wachstum als die Aussicht auf extreme Profite, die junge Unternehmen dazu veranlassen, sich auf diesem Markt zu etablieren. (Zahlen aus England laut NCC Microsystems Centre).

Vertrieb und Marketing

Marketing und Vertrieb werden häufig verwechselt: Der Vertriebsmann ist für den direkten Kontakt zum Kunden zuständig, während Marketing eher eine strategische Aufgabe ist: Die Verpackung des Produktes muß geplant werden, Namen müssen gefunden werden, und auch die Werbung bedarf sorgfältiger Planung.

In der Microcomputer-Industrie gibt es zwei Hauptbranchen – Hersteller und Wiederverkäufer. Vertriebsleute von Hard- und Softwareherstellern verkaufen hauptsächlich an Händler oder Handelsketten. Das Vertriebspersonal von Wiederverkäufern (Systemhäuser, Einzelhandel) wendet sich direkt an den Endabnehmer. Die Verkaufsgewohnheiten sind national sehr verschieden – während in den USA der schnelle Verkauf über den Ladentisch die Regel ist, werden in Europa meist noch Vorverhandlungen vereinbart. Oft geht einem Verkaufsabschluß auch eine Mailing-Aktion oder ein telefonisches Verkaufsgespräch voraus.

Zu Beginn der Computer-Ära wurde das Marketing noch recht stiefmütterlich behandelt: Es wurde planlos gefertigt und verkauft. Heute hat man dazugelernt. Der erfolgreichste Heimcomputer-Hersteller war 1985 die Firma Amstrad (in der BRD Schneider). An der Technologie gab es nichts Neues zu entdecken, das Design der Geräte entsprach jedoch den Wünschen der Abnehmer. Grundsätzlich ist das Erfassen vorhandener Wünsche eine der Hauptaufgaben des Marketing.

ten viel verdient, es können aber wichtige Erfahrungen gesammelt werden. Danach befindet man sich in einer besseren Position, wenn es an die Verhandlungen mit einem professionellen Computer-Handelsunternehmen oder gar einem Hersteller geht. Aber auch bei einer erfolgreichen Bewerbung bleibt die Bezahlung im ersten Jahr niedrig. Der Neuling wird als Vertriebsassistent einem erfahreneren Verkäufer zugeordnet und muß zuerst einmal lernen, wie Computer oder Software-Pakete vorgeführt werden.

Der Weg nach oben

Wer sich bewährt, hat gute Aufstiegsmöglichkeiten. Einjährige Erfahrung bei einer der größeren Firmen genügt oft schon für den Einstieg als „richtiger" Vertriebsmann, der bereits 30–40 000 Mark verdienen und diese durch Provisionen theoretisch noch verdoppeln kann. Im allgemeinen ist auch noch ein Firmenwagen für Außendienste drin.

Innerhalb von drei bis vier Jahren kann aus dem Anfänger ein Profi werden, dem auch wichtige Kunden anvertraut werden. Ein Unternehmen, das sowohl Ein- wie auch Mehrplatzsysteme verkauft, überläßt die teureren Anlagen natürlich den erfahrenen Mitarbeitern, und damit haben diese auch höhere Provisionseinnahmen. Der Verdienst kann jetzt bereits zwischen 50 000 und fast 100 000 Mark liegen, wenn der Erfolg beim Verkauf gut ist. Nach dem Erreichen des 30. Lebensjahres sollte versucht werden, auf den Sessel eines Vertriebsmanagers zu kommen. Die Arbeit bekommt organisatorischen Charakter: So müssen etwa die Vertriebsziele für die Mannschaft festgelegt werden. Wegen der Nähe zum Marketing-Bereich wird dieser Zeitpunkt oft gewählt, um in dieses Tätigkeitsfeld überzuwechseln.

Man darf nie vergessen, daß der Vertrieb ein unsicheres Gewerbe ist, speziell in der risikoreichen Computer-Branche. Nicht selten sind die Vertriebsziele kaum zu erfüllen. Wer mehrere Quartale unter dem Soll bleibt, findet sich leicht auf der Straße wieder. Für Sicherheitsfanatiker ist diese Arbeit ganz bestimmt nicht das Richtige!

Auch beim Marketing ist die Konkurrenz groß. Zwar geben sich manche Firmen mit einem Fachhochschul-Diplom zufrieden, überwiegend wird jedoch ein Universitätsabschluß gefordert. Besonderer Wert wird auf gute Noten in naturwissenschaftlichen Fächern wie

Kettenreaktion

Das britische Unternehmen Granada Business Centres in England ist eine stark expandierende Kette von Computer-Händlern, die sich auf den Verkauf von Ein- und Mehrplatzrechnern an kleine und mittelgroße Unternehmen spezialisiert hat. Als Grundvoraussetzung einer Neueinstellung nennt man bei Granada Verkaufserfahrung im gewerblichen Bereich, noch lieber wird Erfahrung im Verkauf von Computersystemen gesehen. Granada bildet seine zwischen dem 23. und dem 30. Lebensjahr eingestellten Vertriebsleute selbst aus. In jedem Geschäft gibt es einen Vertriebsbeauftragten für Großkunden und einen Vertriebsleiter, der die Aktivitäten des betreffenden Geschäftes überwacht. Ein Bezirksleiter ist für jeweils vier Geschäfte und die dazugehörige Servicemannschaft zuständig.

Die Marketingabteilung bei Granada ist klein. Auf die 60 Vertriebsleute kommen nur vier Marketingmanager, von denen jeder einen Bereich bearbeitet: Der eine ist für Ausstellungen zuständig, der nächste für Pressearbeit, der dritte für die Gestaltung der Läden und so weiter.

Das kleine Wunder

Psion ist ein englisches Unternehmen, das sich mit Software-Entwicklung und dem Bau von Handheld-Computern befaßt. Die Firma hat 70 Angestellte. Von Neulingen im Vertriebsbereich werden zwar Erfahrungen im Verkauf von technischen Geräten oder aus dem Bürofachhandel erwartet, spezielles Computerwissen setzt man jedoch nicht voraus. Psion schult seine Angestellten an den eigenen Produkten. Der Vertriebsmann arbeitet direkt mit Psion-Händlern und größeren Einkaufsgemeinschaften zusammen.

Ein Vertriebsbeauftragter ist bereits für einen festen Kundenstamm zuständig und überwacht die Arbeit der beigeordneten Assistenten.

Bei Psion werden nur Bewerber mit Hochschulabschluß im Marketing beschäftigt. Sie erstellen die Entwürfe für Werbung und Handbücher, organisieren Messestände und bearbeiten Anfragen der Presse.

Marketingleiter und Direktor sind für langfristige Planungen zuständig, sie instruieren etwa die Werbeagentur und lassen Marktforschung durchführen.

Kleine Unternehmen können oft flexibler sein. Sie bieten auch Chancen für solche Interessenten, die sich über ihren Berufsweg noch nicht klar sind. So stellte Psion 1983 etwa einen Programmierer ein, der später zur technischen Vertriebsberatung überwechselte und kurz darauf Vertriebsbeauftragter wurde. Ende 1985 war er dann in der Exportabteilung gelandet.

Marketing beinhaltet erheblich mehr als nur den Verkauf eines Produktes. Wer im Marketing tätig ist, begleitet eine Neuentwicklung vom Konzeptions- und Entwicklungsstadium an. Marketing-Fachleute müssen aufmerksam darüber wachen, daß neue Geräte nicht an den Bedürfnissen des Endabnehmers vorbeigeplant werden. Ein Beispiel für gutes Marketing bietet die Firma Schneider. Obwohl jeder ihrer Computer mit im Grunde bereits überholter Technik arbeitet, kann sie doch den Wunsch des Verbrauchers nach preisgünstigem Computereinsatz voll abdecken.

Mathematik oder Physik gelegt.

Eine Besonderheit des Marketing-Bereiches ist es, daß die Arbeitsprinzipien völlig unabhängig vom Produkt sind: Wer im Marketing von Waschpulver Erfolge vorweisen kann, hat auch in der Computer-Branche Chancen. Viele kleinere Computer-Firmen verfügen nicht über die Mittel, ihre Marketing-Spezialisten selbst auszubilden, zum Vorteil für denjenigen, der bereits zwei bis drei Jahre unter der Obhut eines Großunternehmens zugebracht hat.

Überzeugungsarbeit

Computer-Kenntnisse sind im Marketing kein Muß – aber natürlich ist Interesse an der Sache kein Fehler. Ein Marketing-Assistent mit ein- bis zweijähriger Berufserfahrung verdient ca. 40 000 Mark jährlich. Seine Aufgabe besteht darin, beispielsweise Handbücher und Prospekte zu texten, die Händler mit Informationen zu versorgen, Seminare und Pressekonferenzen abzuhalten und auch Kontakte zu Journalisten zu pflegen.

Der nächste Schritt führt zur Position des Marketing-Leiters. Er hält den Kontakt zur Werbeagentur, entscheidet über Messeaktivitäten und hat auch bei der Einführung neuer Produkte und der Planung einer Firmenstrategie bedeutenden Einfluß. Das Einkommen eines Marketingleiters überschreitet meist die 60 000 Mark, dazu wird er im allgemeinen auch mit einem Firmenwagen und anderen Extras ausgestattet. In kleinen Firmen bleibt dann nur noch der Aufstieg zum Marketing-Direktor.

Im Großformat

Obgleich die Lücke zwischen Micro- und Minicomputern kleiner wird, kommen die Micros an die Leistungsfähigkeit ihrer größeren Brüder noch nicht heran. Als Beispiel für ein bewährtes Minicomputersystem stellen wir die VAX 11/780 vor.

Angesichts der Möglichkeiten moderner Microcomputer mit ihrer Vernetzbarkeit und ihrem vielseitigen Softwareangebot ist die Frage naheliegend, weshalb sich die großen und kostspieligen Minicomputersysteme von Firmen wie DEC oder Prime überhaupt noch verkaufen lassen. Täten es ein paar IBM PCs mit entsprechender Plattenkapazität nicht auch? Wohl kaum, denn das Leistungsvermögen der Micros reicht an das eines mittleren Minicomputers bei weitem nicht heran; der Abstand in Rechengeschwindigkeit, Speicherkapazität sowie Soft- und Firmware-Angebot ist nach wie vor enorm.

Zu den bekanntesten Minicomputern gehören die VAX-Rechner von DEC (Digital Equipment Corporation). Die VAX-Familie umfaßt Systeme unterschiedlicher Leistung, von der MicroVAX, die nicht wesentlich größer als ein üblicher Bürorechner ist, bis zum neuen Spitzenmodell 8800. Dieser Artikel beschäftigt sich mit der schon länger angebotenen „mittleren" VAX 11/780, deren Architektur sich aber nicht grundsätzlich von der anderer VAX-Versionen abhebt.

Schon äußerlich besteht keine Gefahr, die 11/780 mit einem Micro zu verwechseln: Die Maschine beansprucht einen 1 m breiten, 1½ m hohen und ¾ m tiefen Schrank. Und während die meisten kleineren Computer sich mit einer einzigen Leiterplatte, allenfalls noch einer zusätzlichen Erweiterungskarte begnügen, sind bei der 11/780 allein 16 Platinen für Speicherchips vorgesehen, abgesehen von den CPU- und Interface-Karten.

Auch bei der CPU ist es nicht mit einem Chip oder einer Leiterplatte getan. Alle VAX-Rechner verfügen als 32-Bit-Maschinen über einen Adreßraum von vier Gigabyte (vier Milliarden Byte), und eine solche RAM-Kapazität wäre trotz des Preisverfalls bei den Chips weder zu bezahlen noch unterzubringen. Daher wird zur vollen Nutzung der Adressierungsmöglichkeiten ein „virtueller" Speicher organisiert: Nur ein Bruchteil der vier Gigabyte liegt tatsächlich als RAM vor, der Löwenanteil dagegen als Platten- oder Bandspeicher. Hardwaremäßig wird jeweils geprüft, ob ein angefordertes Byte sich bereits im RAM befindet; andernfalls wird die entsprechende „Page" (Seite) automatisch vom Externspeicher geladen.

Das „Paging"-Verfahren beruht auf einer festen Speichereinteilung in Seiten zu 512 Byte. Ist das verfügbare RAM bei Abruf neuer Seiten voll belegt, werden bereits bearbeitete wieder peripher ausgelagert.

Eine Nummer größer

Die VAX-Rechner laufen unter dem DEC-Betriebssystem VMS (Virtuelles Mehrbenutzer-System) oder der Unix-Version „Ultrix 32". Das VMS verfügt über eine leistungsfähige Kommandosprache und einen großen Vorrat an Dienstprogrammen wie etwa Editierhilfen, Sortieralgorithmen und Routinen für die Datenfernübertragung. Als Sprache wird lediglich ein Makro-Assembler mitgeliefert, es sind jedoch Compiler für rund zwanzig Hochsprachen verfügbar. Der VAX-Assembler ist für Programmierer, die den Z80- oder 6502-Code kennen, so etwas wie eine Offenbarung. Die 11/780 bietet z. B. 16 Register zu 32 Bit, die bis auf vier reservierte (für Stack Pointer, Befehlszähler usw.) frei verwendbar sind. Und während Micros intern meist nur ganzzahlig rechnen können, gibt es hier Maschinenbefehle für eine 64-Bit-Gleitkomma-Arithmetik, außerdem für die Zeichen-Manipulation. Zu dem kompletten, 248 Kommandos umfassenden Befehlssatz ge-

Die Kosten für den VAX-Minicomputer 11/780 von DEC dürften wohl nur in größeren Firmen locker zu machen sein. Eine derartige Investition zahlt sich aber aus, wenn kontinuierliche Massenspeicherung von Daten, hohe Verarbeitungsgeschwindigkeit sowie große Flexibilität gefordert sind.

hören auch die üblichen Schiebe- und Vergleichsoperationen sowie eine Reihe spezieller Anweisungen für die CPU-Steuerung.

Um am VAX-Betrieb teilzunehmen, benötigt man zunächst für das Einloggen vom Systemoperator eine Benutzerkennung und ein Paßwort. Damit haben Sie Zugang zu bestimmten Systembereichen; an diese Kennung sind individuelle Befugnisse und auch Einschränkungen gebunden, so daß Sie nicht auf jede Datei und jede Systemfunktion zugreifen können. Genauso lassen sich eigene Daten bei Bedarf gegen unberechtigtes Lesen oder Überschreiben schützen.

Kommandoprozeduren

Die VAX-Kommandosprache DCL (Digital Command Language) bietet viele Möglichkeiten, die sonst nur in Hochsprachen zu finden sind. Sie können Variablennamen einführen, GOTO- und IF-Anweisungen ähnlich wie in BASIC verwenden und außerdem über eine Reihe „lexikalischer" Funktionen Systeminformationen wie Rechenzeiten und aktuelle Benutzernamen abfragen. Die meisten dieser Befehle finden nur in „Kommandoprozeduren" Verwendung; das sind Sequenzen von DCL-Befehlen, die wie ein Programm abgearbeitet werden. Für viele Zwecke reicht aber auch schon ein einziges Kommando. „Show Users" beispielsweise veranlaßt die Ausgabe einer Benutzerliste. Lexikalische Funktionen innerhalb von Kommandoprozeduren erlauben es u. a., an jeden Benutzer eine Mitteilung zu senden.

Bei der Mehrzahl der Befehle gibt es verschiedene Modifikationen. So löscht DELETE*.* sämtliche Dateien einer Directory, soweit sie nicht geschützt sind, während DELETE/CONFIRM*.* zusätzlich für jedes File eine Löschfreigabe verlangt. Die vollständige Befehlsliste ist sehr umfangreich und enthält neben den üblichen Kommandos für den Umgang mit Dateien auch solche zur Umgestaltung der Programmierumgebung und zur Kommunikation mit anderen Benutzern.

Die Verbindung des Rechners mit der Außenwelt erfolgt über zwei Schnittstellen: Der „Unibus" (ein DEC-Standard) dient zum Anschluß von Terminals, langsamen Platten- und Bandspeichern sowie Druckern und einer Vielzahl von Spezialgeräten wie Plottern und Steuerungen, die zum DEC-Programm gehören. Sehr viel höhere Übertragungsraten (im Bereich von 10 MByte/s) gestattet der „Massbus", der beispielsweise für Plattenlaufwerke verwendet wird.

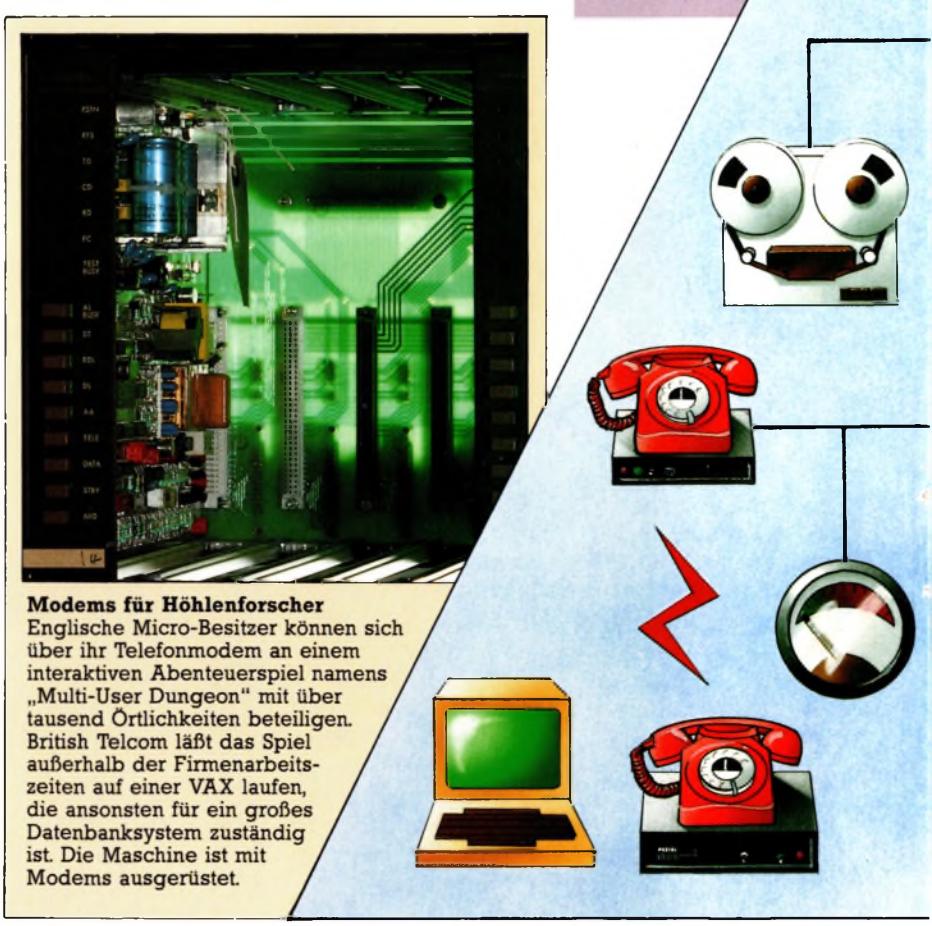
Die kleinste 11/780-Konfiguration verfügt über zwei MByte RAM und acht Terminalanschlüsse. Für einen sinnvollen Betrieb sind zumindest noch zwei Plattenlaufwerke mit einer Minimalkapazität von je zwei MByte erforderlich. Die Anlage läßt sich auf acht Laufwerke und einen Arbeitsspeicher von acht MByte erweitern, durch Einbau zusätzlicher Interface-Karten außerdem auf 64 Terminals.

Das VMS-Betriebssystem eignet sich hervorragend für die Programmentwicklung: Die Kommandosprache ist logisch strukturiert, und für alle Befehle gibt es jederzeit aufrufbare Help-Files. Ferner sind praktische Bibliotheksroutinen für die Übernahme in eigene Programme verfügbar. Daher entfällt weitgehend die Notwendigkeit, Dinge wie Bildschirmmasken oder mathematische Algorithmen selbst zu programmieren. Auch erspart die gemeinsame Benutzung der gleichen Plattendateien durch die Angehörigen von ganzen Software-Mannschaften all den Ärger, der bei der Programmentwicklung auf getrennten Microcomputern dadurch entsteht, daß jemand in seinen Dateien Änderungen vornimmt, ohne die anderen zu informieren.

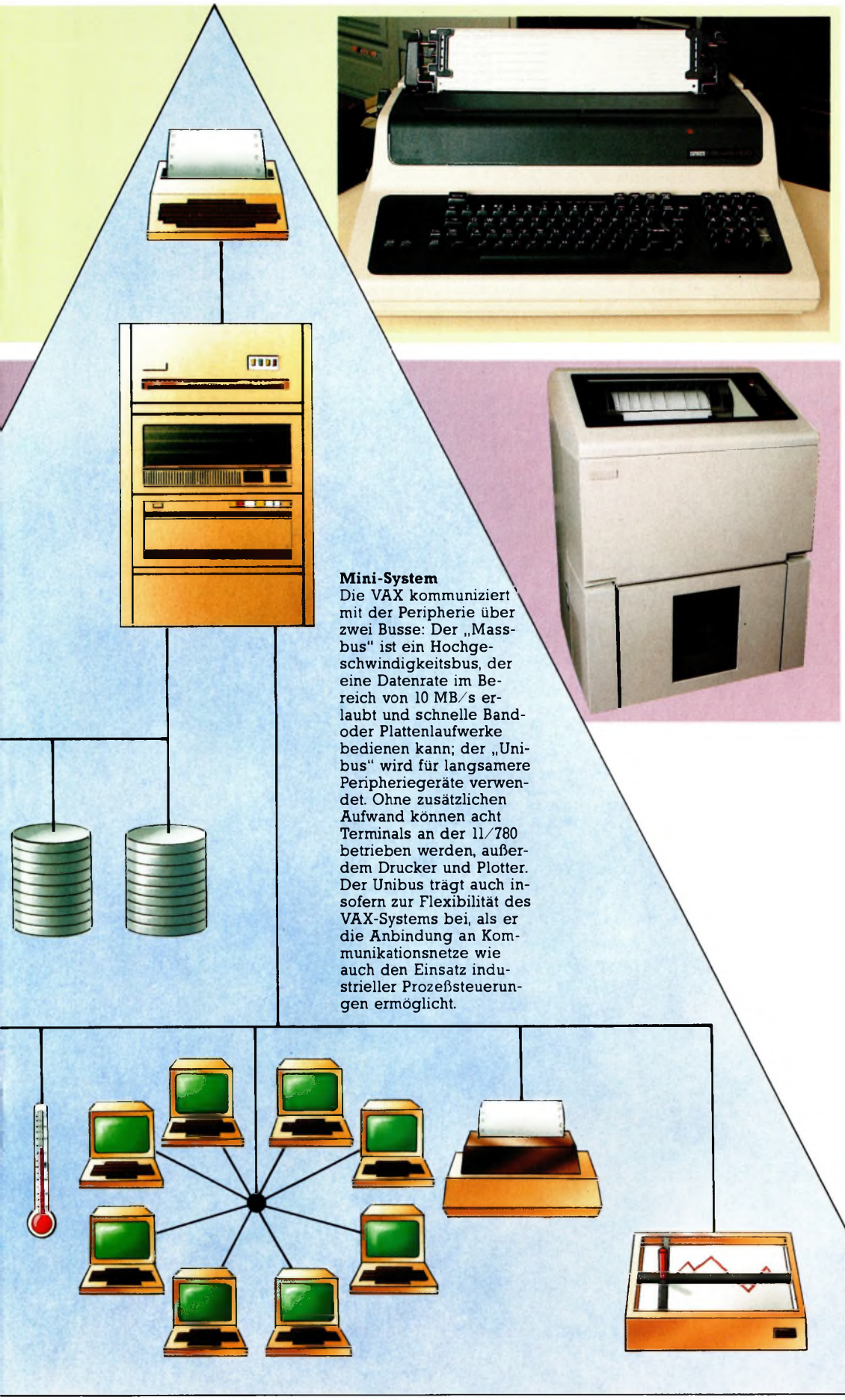
Insgesamt bietet ein derartiger Minicomputer eine sehr benutzerfreundliche Programmierumgebung, und dank des hohen Datendurchsatzes laufen auch aufwendige Programme wesentlich schneller als auf jedem Micro. Das hat natürlich seinen Preis: Die VAX 11/780 kostet in Standardausführung schon so viel wie eine Villa in bester Wohnlage. Auch die laufenden Kosten sind wegen des elektrischen Anschlußwerts von 6 kW zuzüglich Klimaanlage und wegen der Wartungsgebühren nicht unerheblich. Dafür wird perfekter Service garantiert.

Spitzenposition
Der Systemoperateur der VAX-Anlage verkehrt mit dem Computer über den Konsolschreiber rechts im Bild, der wie eine Fernschreibmaschine betrieben wird und auch die Dialogantworten des Systems oder Zustandsmeldungen ausdruckt. Auf diese Weise entsteht ein vollständiges Protokoll der Zwiesprache des Operators mit der betriebenen Anlage.

Zeilendrucker
Über den DEC-Unibus kann an die VAX ein schneller Zeilendrucker angeschlossen werden, der pro Sekunde an die zwanzig Zeilen mit 132 Zeichen zu Papier bringt.



Modems für Höhlenforscher
Englische Micro-Besitzer können sich über ihr Telefonmodem an einem interaktiven Abenteuerspiel namens „Multi-User Dungeon" mit über tausend Örtlichkeiten beteiligen. British Telcom läßt das Spiel außerhalb der Firmenarbeitszeiten auf einer VAX laufen, die ansonsten für ein großes Datenbanksystem zuständig ist. Die Maschine ist mit Modems ausgerüstet.



Mini-System
Die VAX kommuniziert mit der Peripherie über zwei Busse: Der „Massbus" ist ein Hochgeschwindigkeitsbus, der eine Datenrate im Bereich von 10 MB/s erlaubt und schnelle Band- oder Plattenlaufwerke bedienen kann; der „Unibus" wird für langsamere Peripheriegeräte verwendet. Ohne zusätzlichen Aufwand können acht Terminals an der 11/780 betrieben werden, außerdem Drucker und Plotter. Der Unibus trägt auch insofern zur Flexibilität des VAX-Systems bei, als er die Anbindung an Kommunikationsnetze wie auch den Einsatz industrieller Prozeßsteuerungen ermöglicht.

DEC VAX 11/780

ABMESSUNGEN:
1181 × 1537 × 762 mm

ZENTRALEINHEIT:
VAX-CPU von DEC

SPEICHER:
2–8 MByte RAM, bis zu 4 Gigabyte virtueller Speicher adressierbar

BILDSCHIRM:
Mindestens 25 Textzeilen zu 80 Zeichen; hochauflösende Grafik mit bis zu 1024 × 1024 Pixeln bei über 16 Millionen Farbtönen (abhängig von der Terminal-Ausführung)

SCHNITTSTELLEN:
Massbus für schnelle Platten- und Bandspeicher; Unibus für alle anderen Geräte einschl. Drucker, Plotter und Terminals (über RS232-Interface)

VERFÜGBARE SPRACHEN:
Assemblerpaket wird mitgeliefert, Compiler für fast alle gängigen Hochsprachen erhältlich

DOKUMENTATION:
Die Grundausstattung an System-Handbüchern einschl. der Dokumentation für die Kommandosprache und die Dienstprogramm-Bibliothek umfaßt 30 Bände und belegt ein Zweimeter-Regal; inbegriffen sind ein knappes Nachschlagewerk und Benutzer-Einweisungen.

STÄRKEN:
Die VAX bietet eine äußerst produktive Umgebung für die Software-Entwicklung und die Datenbankorganisation. Auch die Vernetzung von VAX-Rechnern ist vorgesehen.

SCHWÄCHEN:
So ein System ist nicht gerade billig und verursacht zudem hohe laufende Kosten, weil Raumklimatisierung, Luftfilterung und der Abschluß eines Wartungsvertrags unumgänglich sind.

Unter der Haube

Unsere Hardware-Serie wird mit der Betrachtung von drei Heimcomputern abgeschlossen. Dabei zeigen sich individuelle Unterschiede in der Architektur. Anhand vereinfachter Logik-Diagramme wird außerdem gezeigt, wie das Innenleben der meistbenutzten Gatterbausteine aussieht.

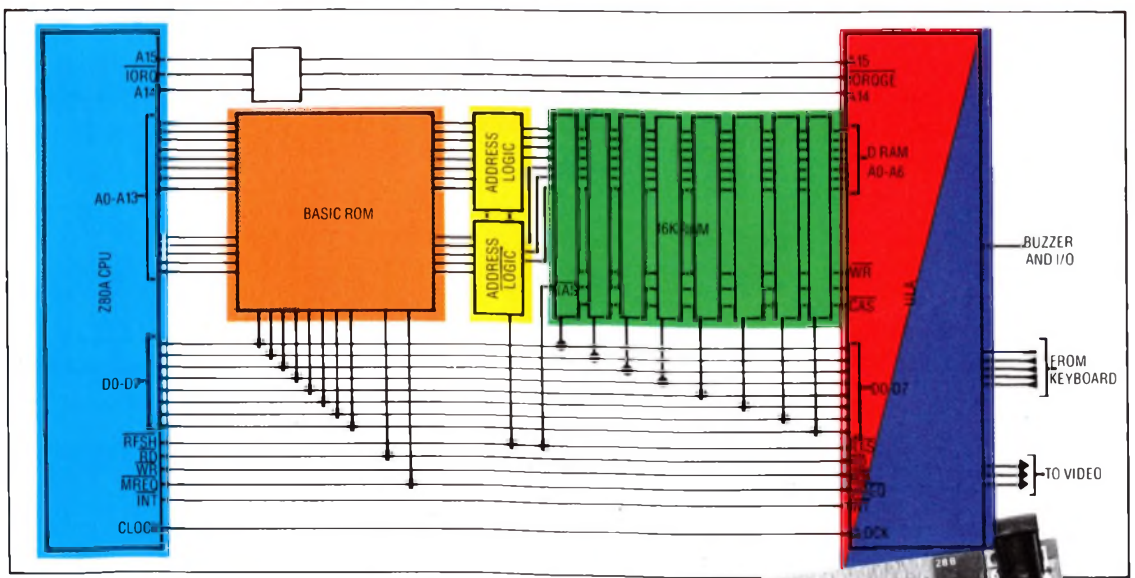
Sinclair Spectrum

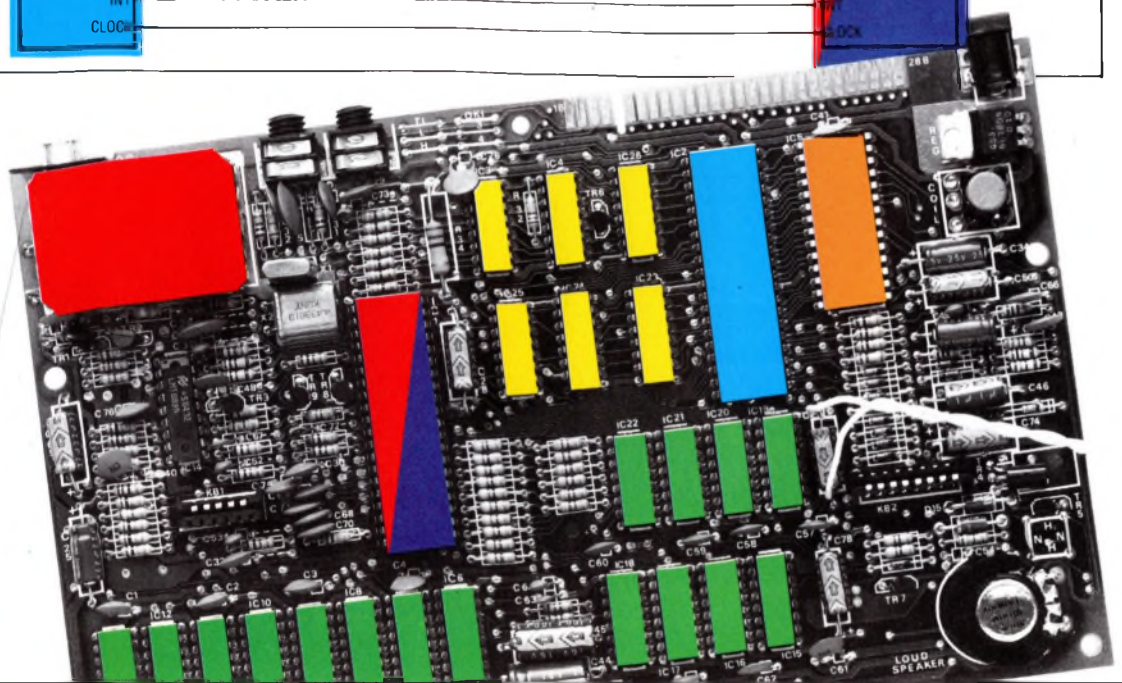
Der ZX Spectrum weicht in seiner Architektur erheblich von den anderen hier besprochenen Micros ab. Als CPU dient ein Z80A; für die meisten Systemfunktionen ist ein ULA (Uncommitted Logic Array) zuständig. Dieser maßgeschneiderte Chip steuert die Videosignale, die Speicherauffrischung und die elementare Ein/Ausgabe über Tastatur und Cassettenrecorder sowie den Minilautsprecher. Das untenstehende Blockdiagramm gilt für die ältere 16-KByte-Version, die späteren 48K-Modelle enthalten zusätzliche RAM-Bausteine.

Die Bildschirmwiedergabe erfordert in regelmäßigen Abständen schnelle Speicherzugriffe. Die ersten 16 K RAM mit der Videoinformation haben beim 48K-Spectrum direkte Verbindung mit dem ULA-Baustein, der deshalb unter normalen Umständen das Video-RAM unabhängig von der CPU ansprechen kann. Diese kann gleichzeitig mit dem BASIC-ROM oder den 32 K Zusatz-RAM kommunizieren. Kritisch wird es, wenn auch die CPU auf den 16K-Block mit dem Video-RAM zugreifen möchte, weil die dort ebenfalls gespeicherten BASIC-Systemvariablen benötigt werden.

Farbschlüssel

- Prozessor
- Dynamisches RAM
- ROM
- Ein/Ausgabe
- Videosignal-Erzeugung
- System- bzw. Adreßlogik



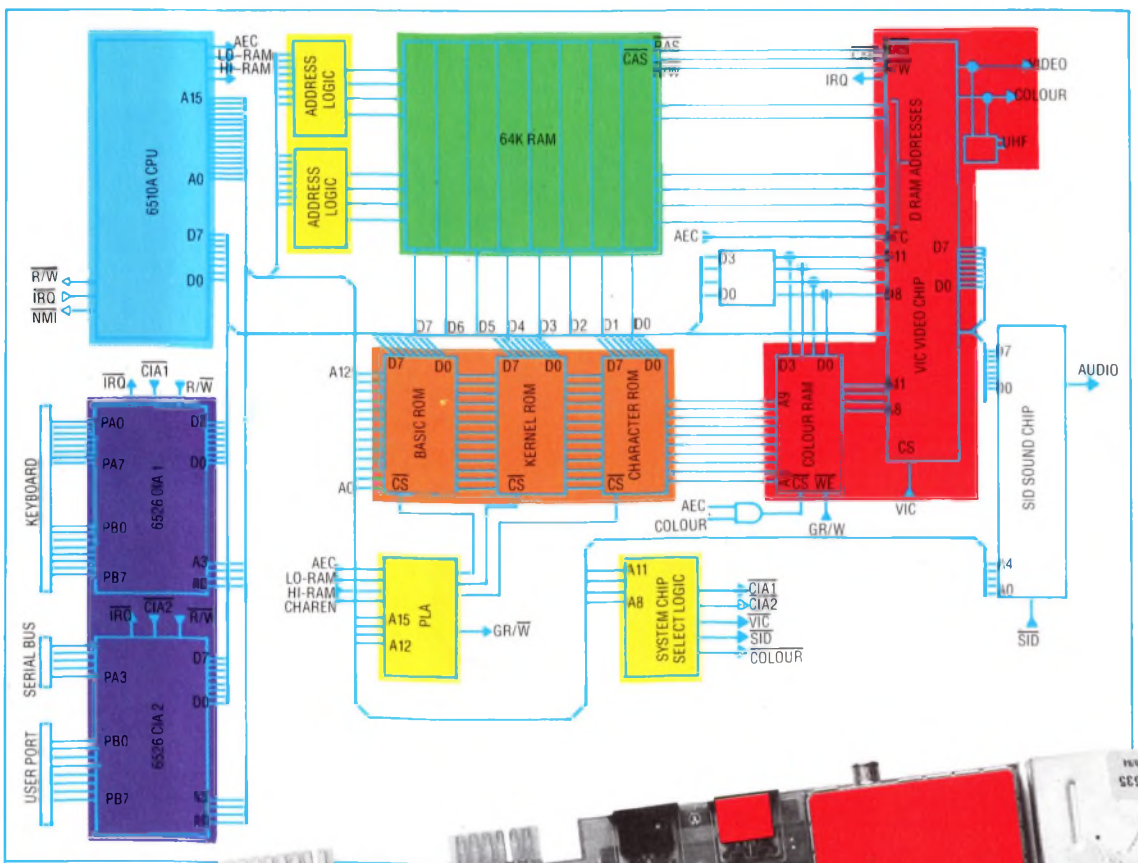


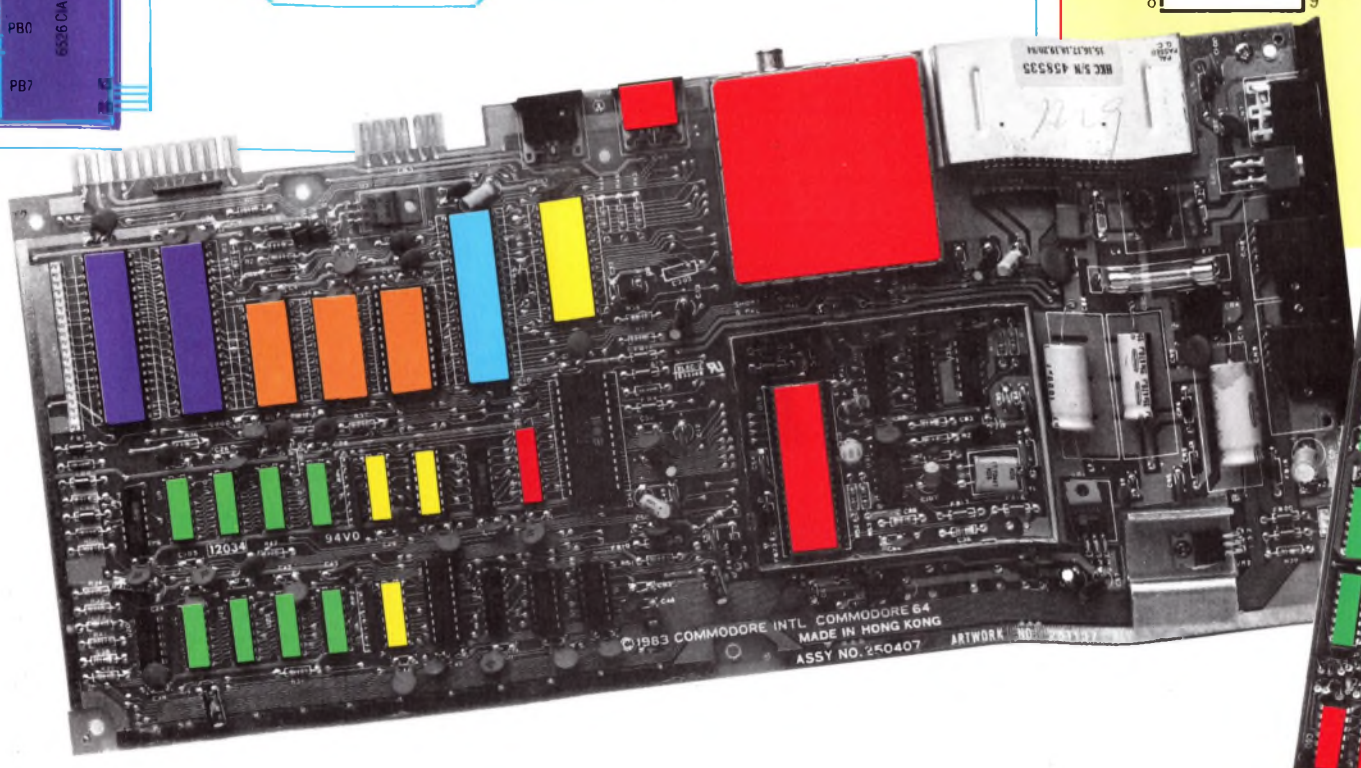
Commodore 64

Der C 64 ist recht gut ausgestattet, mit 64 KByte RAM (dynamisch) in acht 64 KBit-Chips sowie mit drei 8 KByte-ROMs, die den BASIC-Interpreter, den Systemkern (Kernel) und den Zeichensatz (Character ROM) enthalten. Die CPU 6510 ist eine verbesserte 6502-Version, die das Einblenden der ROMs in den Adreßraum des Prozessors über ein integriertes Zusatzregister und einen speziellen PLA-Chip (Programmed Logic Array) ermöglicht.

Der C 64 hat zwei Schnittstellenbausteine (PIO = Peripheral Input/Output), einen davon für die Tastatur. Beim zweiten Chip wird ein Port für den seriellen Commodore-Bus verwendet, an dem Drucker, Diskettenlaufwerke und andere Peripheriegeräte hängen, der andere steht als User Port zur Verfügung.

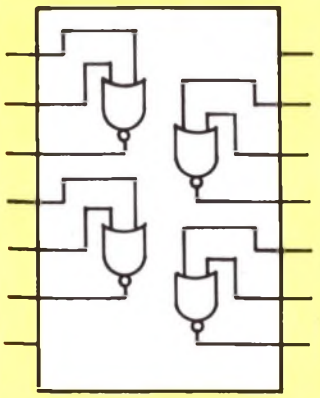
Der Video-Chip (VIC = Video Interface Controller) holt sich die Bildschirminformation direkt aus dem dynamischen RAM. Auch hier muß der Videochip die CPU anhalten, wenn er auf das RAM zugreifen will.



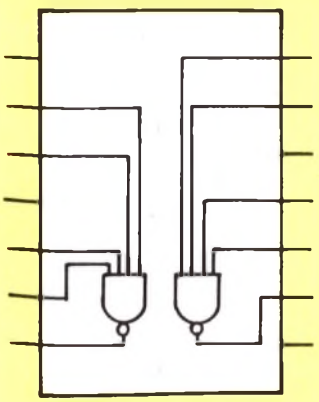


Ganz logisch

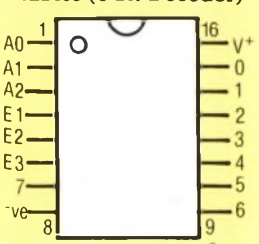
74LS00 (Vierfach-NAND mit je zwei Eingängen)



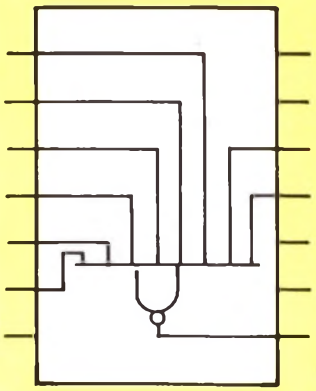
74LS20 (Zweifach-NAND mit je vier Eingängen)



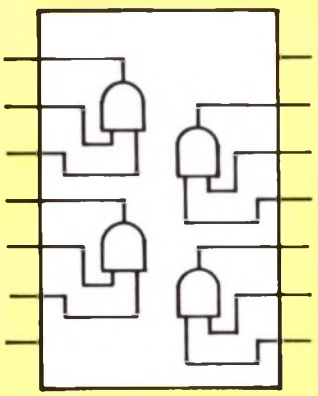
74LS138 (3-Bit-Decoder)



74LS30 (Einfach-NAND mit acht Eingängen)



74LS02 (Vierfach-NOR mit je zwei Eingängen)

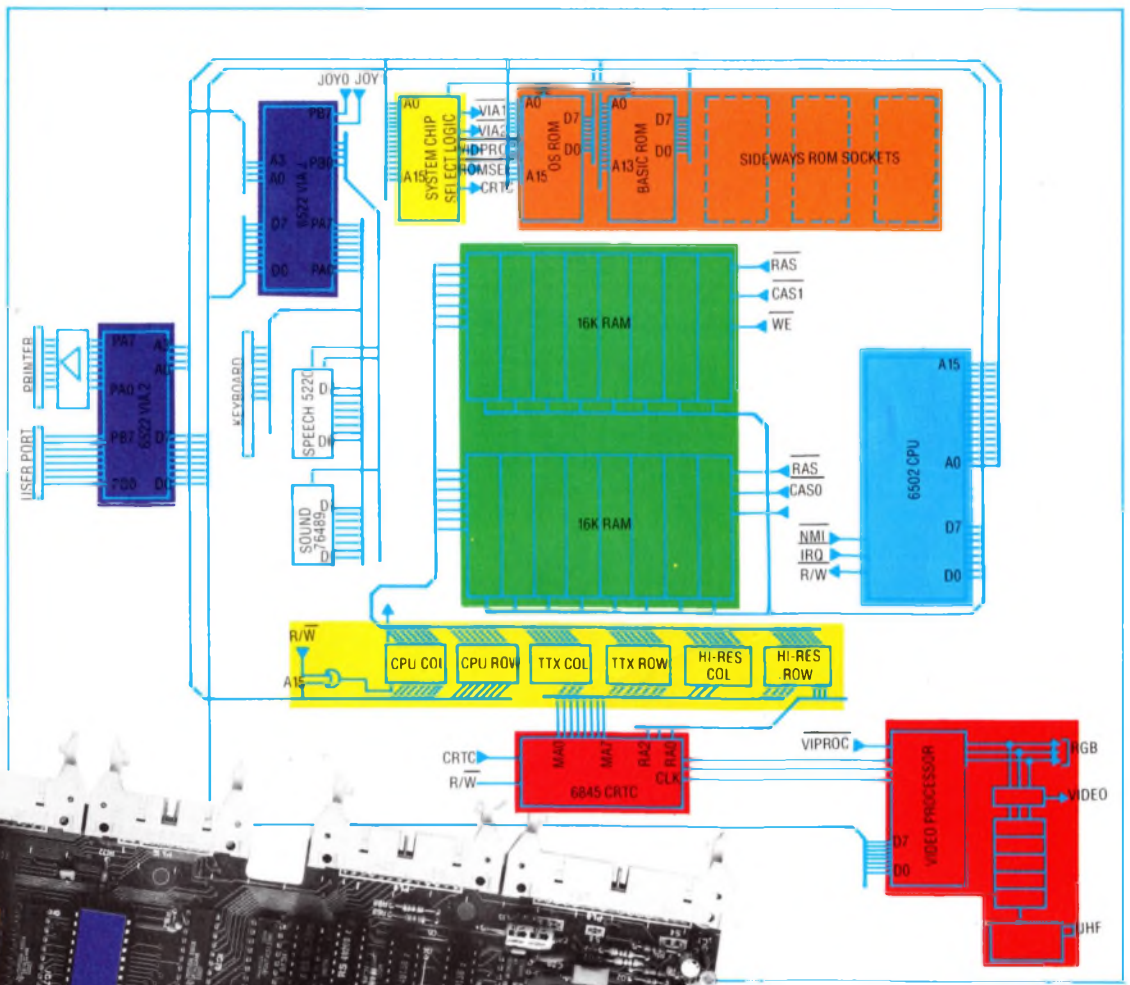


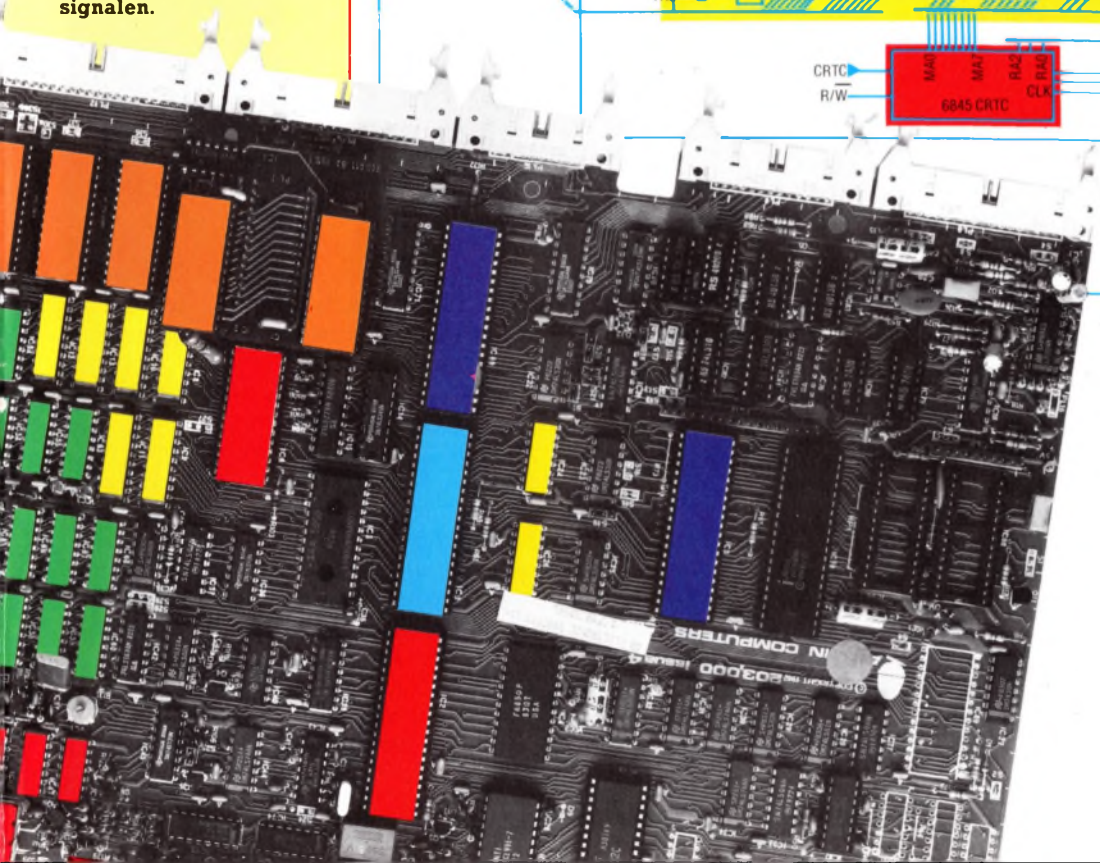
Von den kleineren Bausteinen auf einer Microcomputer-Platine stammen viele aus der 74er-Serie. Sie enthalten diverse Gatterkombinationen und dienen zur Verknüpfung oder Decodierung von Adreß- und Steuersignalen.

Acorn B

Der Acorn B bietet vielseitige Möglichkeiten. Dennoch ist der Platine anzusehen, daß sie älteren Datums ist: Für 32 KByte RAM sind hier 16 dynamische RAM-Chips mit je 16 KBit nötig, während der Commodore 64 mit nur acht Bausteinen auf 64 KByte kommt. Für den wechselweisen Zugriff der CPU und der Video- bzw. Teletext-Chips auf das RAM sind sechs Adreßlogik-Bausteine vorgesehen. Die Grundausstattung enthält ferner zwei 8-KByte-ROMs und drei Leersockel für weitere ROMs.

Die Platine hat außerdem zwei PIOs und einen seriellen Schnittstellenbaustein. Ein PIO wird vom System für die Tastatur, die Klang- und Sprachsysthese-Bausteine, die Feuerknöpfe der Joysticks sowie für einige Videofunktionen wie das „Rollen" des Bildschirms (Scrolling) verwendet. Über den Port A des System-PIOs ist eine Art langsamer Datenbus eingerichtet; die CPU verkehrt mit der Tastatur und dem Tongenerator nicht unmittelbar. Beim zweiten PIO-Chip ist ein Port für den Paralleldrucker zuständig, der andere dient als User Port.





Computerprodukte

In der Produktion spielen Computer heute eine sehr wichtige Rolle – nicht nur bei Planung und Entwurf: Rechner steuern heute den gesamten Ablauf bis zur Lieferung an den Endverbraucher.

Um effektiv produzieren zu können – von der Entwicklung eines neuen Produktes bis zu seiner Lieferung –, muß eine Vielzahl von Faktoren abgeglichen werden: Termine, Personal, Maschinen und Rohmaterial, Design und Lieferfristen sowie Dutzende anderer Bedingungen. In den frühen 80er Jahren waren nur die größten und teuersten Computer leistungsfähig genug, um erfolgreich in der Wirtschaft eingesetzt zu werden. Hier hat sich inzwischen ein Wandel vollzogen: Zum einen wurden die Computer leistungsfähiger, zum anderen sind viele Unternehmen erst heute auf die Möglichkeiten der Rechner aufmerksam geworden.

In neuerer Zeit trifft man gelegentlich auf Produkte, die ausschließlich mit Hilfe von Computern gefertigt wurden – vom Entwurf bis zur Lieferung an den Endverbraucher. Am weitesten ist die Entwicklung naturgemäß in der Computerbranche selbst fortgeschritten – die Hersteller von Rechnern und elektronischen Bauteilen waren die ersten High-Tech-Unternehmen.

Wenn Computer den gesamten Fertigungsablauf steuern, spricht man von „CIM" (Computer Integrated Manufacturing). Die vielfältigen Abkürzungen können in diesem Bereich allerdings sehr verwirren.

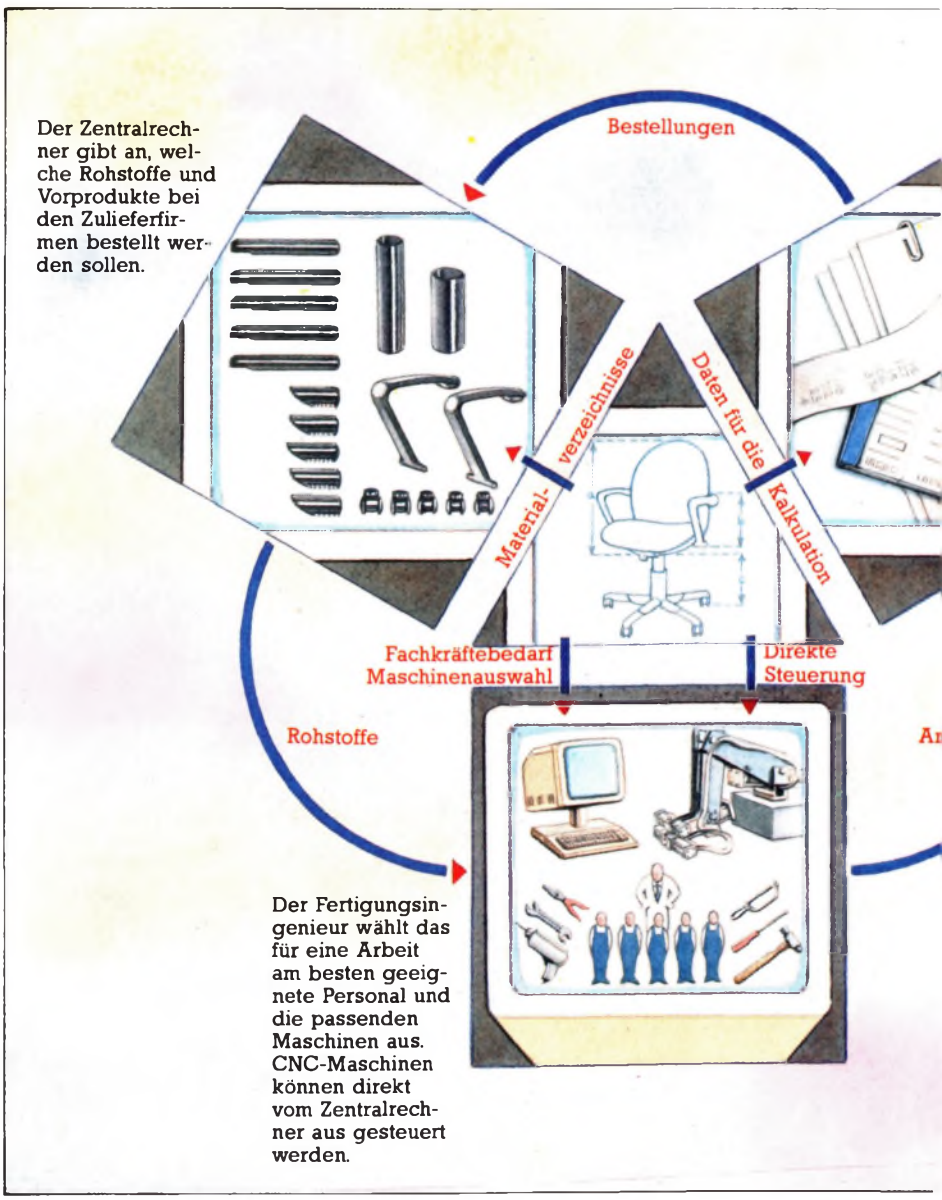
Termine

Große Industriefirmen sind auf Zusammenarbeit angewiesen – jede Abteilung ist für einen anderen Teil der Produktion zuständig. Die Planungsabteilung entwickelt eine Idee und stellt die technischen Spezifikationen zusammen, die zur Basis des gesamten Produktionsprozesses werden. Zur Vorbereitung müssen Bearbeitungswerkzeuge bestellt oder angefertigt werden, Werkzeugmaschinen sind neu zu programmieren. Die Lagerverwaltung muß sicherstellen, daß die nötigen Rohmaterialien oder Vorprodukte jederzeit in ausreichender Menge zur Verfügung stehen.

Dazu kommt die finanzielle Seite des Unternehmens. Bevor ein Produkt bestellt wird, wollen die Kunden natürlich den Preis kennen – der Hersteller muß exakt kalkulieren und genaue Zahlen nennen. Schließlich ist auch die Liefer- und Zahlungsfrist abzustimmen – Waren sollen termingerecht bereitstehen und auch pünktlich bezahlt werden. Die Personalsituation ist ebenfalls zu berücksichtigen: Fachleute mit besonderen Fähigkeiten und Kenntnissen sollen zum richtigen Zeitpunkt für die Arbeit frei sein.

Wie unterschiedlich die vorerwähnten Aufgaben auch immer scheinen mögen, sie basieren alle auf den gleichen Daten. Und das ist der Schlüssel zu CIM, bei dem ein zentraler Computer sämtliche wichtigen Informationen enthält. Alle Abteilungen können darauf zugreifen: Die Lager- und Einkaufsabteilung kann die Produktbeschreibung abrufen und prüfen, welche Materialien gebraucht werden, der Werksleiter sieht, welche Maschinen ein-

Computer können den Fertigungsprozeß viel stärker vernetzen als es früher möglich war. Das Konzept des computergestützten Entwurfs bildet die Basis jeder rechnerorientierten Fabrik, bei der die Elektronik die Steuerung vom Entwurf bis zur Fertigung verwaltet bzw. unterstützt.



Mit der CIM-Datenbank läßt sich ein Zeitplan für die gleichzeitige Fertigung verschiedener Produkte erstellen – die Rohstoffverwaltung, der Maschinen- und Personaleinsatz werden automatisch optimiert. Gleichzeitig informiert das System auch die Buchhaltung über die anfallenden Kosten. Die Menge der notwendigen Rohstoffe kann ebenso genau kalkuliert werden wie die Kosten von Arbeitseinsatz und Maschinenzeit. Aus diesen Informa- gesetzt werden sollen, und die Rechnungsabteilung kann die Kosten kalkulieren – alle Daten sind in einer Datenbank gespeichert.

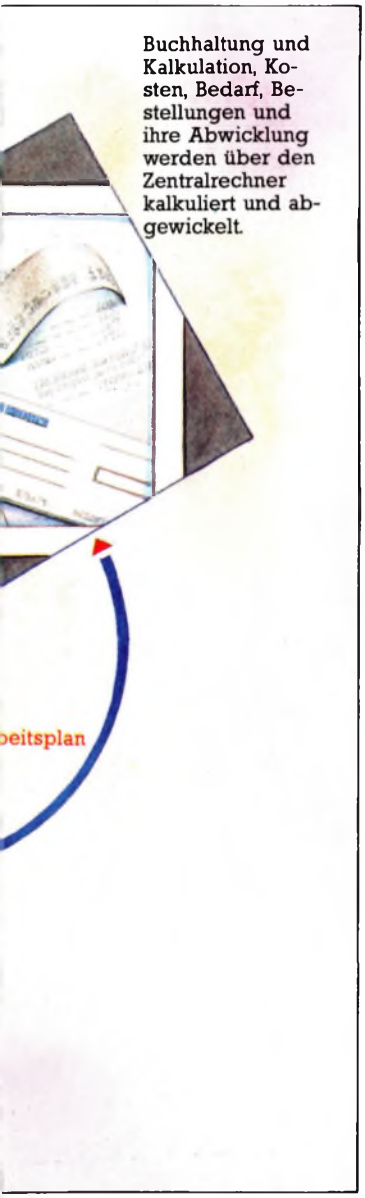
Simulation

Lassen Sie uns die nötigen Informationen genauer betrachten: Am wichtigsten ist natürlich die Beschreibung des Produktes selbst, die von der Planungsabteilung erstellt wird. Dazu werden seit Jahren mit großem Erfolg CAD-Systeme (CAD = Computer Aided Design) verwendet. Nur wenige Erzeugnisse bestehen aus einem einzigen Bauteil, im allgemeinen hat man viele zusammengesetzte Elemente.

Ausgereifte CAD-Systeme können Teil für Teil ein Gesamtbild des Produktes zusammensetzen, wobei Schicht auf Schicht gelegt wird, bis die Gesamtansicht vollständig ist – und das Ganze auch in drei Dimensionen. Aus diesem Bild kann ein geeignetes Programm für Simulation und Analyse bereits die späteren Eigenschaften des Produktes ermitteln.

Das BOM (Bill Of Materials = Materialverzeichnis) ist eine Hilfe für die Lagerverwaltung. Das BOM ist ein Verzeichnis aller Teile eines Produktes und kann wiederum zu einer Liste von Rohmaterialien aufbereitet werden, die für die Herstellung nötig sind. Die Lagerverwaltung muß die vorhandenen Materialien mit dem Bedarf abgleichen. Sie muß auch wissen, was während der Produktion nachzubestellen ist. Die Einkaufsabteilung ist darüber zu informieren, welche Waren von wem zu beziehen sind.

Der CAD-Entwurf dient auch den Fertigungstechnikern – sie erfahren, welche Anforderungen an Maschinen und Personal gestellt werden. Mit diesen Informationen ist es dann ein leichtes, die Anforderungen an Maschinenausrüstung und den Arbeitskräften zu ermitteln und einen Ablaufplan für die Fertigung zu erstellen.



Buchhaltung und Kalkulation, Kosten, Bedarf, Bestellungen und ihre Abwicklung werden über den Zentralrechner kalkuliert und abgewickelt.

Kalkulation

CIM soll aber nicht nur zur Herstellung einzelner Produkte beitragen, sondern die gleichzeitige Fertigung vieler Waren in den unterschiedlichsten Entwicklungsstadien synchronisieren. Situationen, in denen zwar Schrauben fertig sind, die entsprechenden Muttern jedoch nicht, soll es mit CIM nicht mehr geben. tionen kann die für den Kunden wichtige Preisfrage bereits vor Anlaufen der Herstellung genau beantwortet werden.

Später gelangt der im Konstruktionsbüro erstellte Entwurf direkt zu einer rechnergesteuerten Werkzeugmaschine, welche die notwendigen Bearbeitungsschritte einleitet. Auf der Grundlage des Entwurfes hat sich inzwischen ein Netz von Informationen gebildet: Sowohl die Lagerverwaltung, die Fertigung, das Vertriebsbüro und sogar der Versand erhalten ihre Anweisungen und Informationen aus der gleichen Datenbank.

Aus manchen Industriezweigen ist der Computer-Einsatz heute nicht mehr wegzudenken. So ist etwa der Entwurf eines modernen Microprozessors so kompliziert, daß er ohne Rechner nicht mehr zu überblicken ist. Schaltungsentwicklung „per Hand" würde viel zu lange dauern, zudem könnte das von einem Zeichner angefertigte Schaltungslayout nicht einmal getestet werden, ohne den Chip tatsächlich herzustellen. Und das wäre so teuer, daß sich die Produktion nicht lohnen könnte. Um aber effektiv arbeiten zu können, muß die Datenbank erst an die nötigen Informationen kommen.

Industriesprache

Industrieller Computereinsatz wird mit einem Wust von Abkürzungen beschrieben. Einige davon haben wir hier zusammengestellt:

AMT Advanced Manufacturing Techniques – Neue Produktionstechniken: Allgemeiner Ausdruck für CAD, CAM und CAE.

CAD Computer Aided Design – Computergestützter Entwurf: Bezieht sich meist auf das reine Zeichnen mit Computerhilfe. Dabei können zwar Bilder entstehen, analysiert werden sie durch CAD aber meist noch nicht.

CAM Computer Aided Manufacturing – Computergestützte Produktion: Schließt alle Techniken ein, bei denen Rechner in den Produktionsprozeß integriert sind.

CAE Computer Aided Engineering – Computergestützte Analyse: Einsatz der Datenverarbeitung (meist CAD-Datenbanken) für die Analyse eines Entwurfs.

NC Numerical Control – Numerische Steuerung: Datengesteuerte Maschine zur Bearbeitung von Werkstücken.

CNC Computer Numerical Control – Computersteuerung: Weiterentwicklung der NC-Maschinen, die früher noch per Lochstreifen gesteuert wurden. Heute erfolgt die Steuerung direkt über angeschlossene oder integrierte Computer.

CIM Computer Integrated Manufacturing – Computergesteuerte Fertigung: Beschreibt den gesamten Prozeß der automatischen Steuerung von Fertigungsabläufen.

BOM Bill Of Materials – Materialverzeichnis: Liste aller Komponenten, die zu einem Produkt gehören.

Millionen von Schaltern

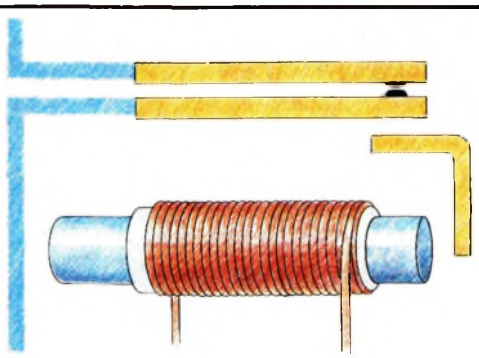
Erst durch die Entwicklung von Transistoren konnten ehemals raumfüllende Apparaturen in handliche Kleincomputer verwandelt werden, die sich jeder leisten kann.

Science Museum

An der Universität von Pennsylvania wurde 1943 für die amerikanische Armee eine Rechenmaschine für die sehr komplexe Berechnung von Geschoßbahnen entwickelt. Die Weltöffentlichkeit erfuhr erst 1946 von dieser Erfindung, die bis zu ihrer Vollendung 7237 Arbeitsstunden benötigte.
Man taufte den Rechner ENIAC (Elektrischer, numerischer Integrator und Kalkulator). ENIAC war der erste Rechner mit Elektronenröhren, und davon benötigte er gleich 18 000 Stück, die eine elektrische Leistung von 200 kW (ungefähr 100 Herdplatten) in abzuführende Wärme umsetzten. Zusätzlich waren 1 500 Relais im Einsatz. Die monströse Konstruktion war in einem Raum von 9 mal 30 Metern untergebracht.

Das Relais als Schalter

Eine Spule mit Eisenkern, von Strom durchflossen, erzeugt an ihren Enden eine magnetische Kraft. Der in seiner rechtwinkligen Biegung gelagerte Eisenblechwinkel wird angezogen. Der umschwenkende Winkel preßt die beiden Kontakte zusammen und schließt den Stromkreis.



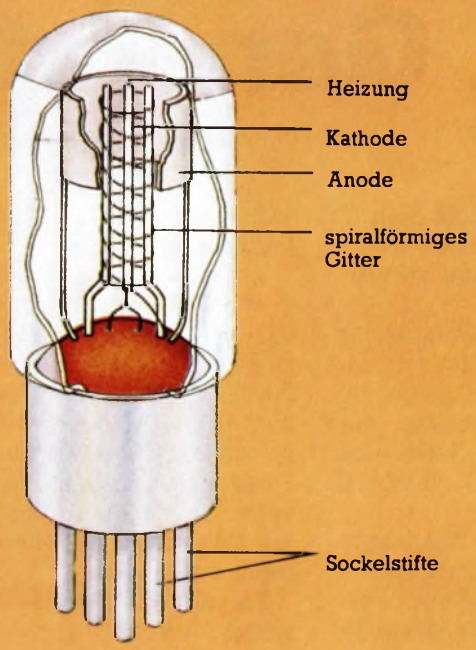
Tony Lodge

Im Computer von heute befinden sich Millionen winziger elektronischer Schalter – die Grundbausteine des Rechners, ohne die die technologische Revolution nicht stattgefunden hätte.

Bereits 1938 demonstrierte der Elektroingenieur Claude Shannon, daß mit elektronischen Schaltern komplizierte logische Operationen durchgeführt werden konnten. Die ersten Konstruktionsversuche führten zum Relais, durch das die ersten Pionierleistungen in der Rechnerentwicklung möglich wurden. Ein Relais

Elektronenröhre

Die Illustration zeigt eine Triode (Röhre mit drei Elektroden), die in einem luftleeren Glaskolben untergebracht ist. Die negative Elektrode, die Kathode, und die positive Elektrode, die Anode, sind durch einen spiralförmigen Draht, das sogenannte Gitter, voneinander getrennt. Die Kathode wird durch einen Strom erhitzt und erzeugt um sich herum eine Elektronenwolke, also negativ geladene Teilchen, die von der Anode angezogen werden. Ein besonderer chemischer Überzug der Kathode fördert die Erzeugung von Elektronen.
Ist am Gitter keine Spannung, behindert es die Elektronenbewegung nicht. Wird eine negative Spannung angelegt, wird der Elektronenstrom zur Anode unterbrochen. Die Röhre hat also die Eigenschaften eines Schalters, der sich durch Anlegen einer negativen Spannung an das Gitter öffnet. Die erste Generation digitaler Computer benutzte Tausende solcher Röhren als Schalter. Die meisten Röhren haben, neben Kathode, Anode und Gitter, zusätzliche Elektroden zur Verbesserung ihrer Arbeitsweise. Die oben beschriebene grundsätzliche Funktion gilt jedoch auch für diese Röhren.
ENIAC konnte nur 20 zehnstellige Zahlen speichern, und programmiert wurde er durch Umstöpseln der vielen Leitungen. Allein während des Jahres 1952 mußten 18 000 Röhren ersetzt werden, weil der Rechner nur ganze zwei Minuten laufen konnte, bis die nächste Röhre durchbrannte. Ende 1952 wurde ENIAC eingemottet.



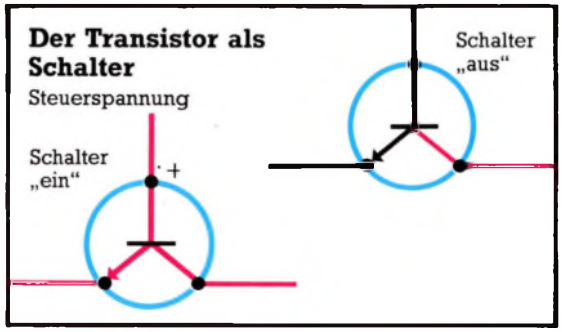
Mullard Ltd

besteht jedoch auch aus beweglichen mechanischen Teilen, so daß die Rechenvorgänge langsam abliefen und häufige Ausfälle das Gerät für den praktischen Einsatz unbrauchbar machten.

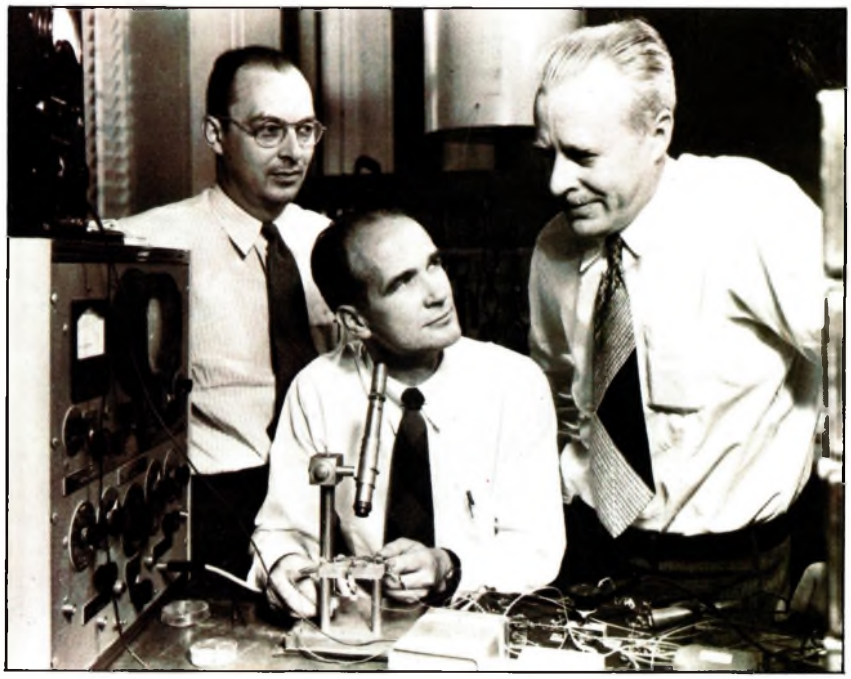
In der ersten Generation praxisgerechter Computer bestanden die Schalter aus Elektronenröhren. Sie arbeiteten rein elektronisch und rechneten um einiges schneller. Der Einsatz von Röhren warf jedoch neue Probleme auf. Der Stromverbrauch und die Wärmeabgabe waren immens, die Rechner benötigten noch immer viel Platz, und die Zuverlässigkeit ließ noch zu wünschen übrig.

Die Erfindung des Transistors bahnte einer neuen Computergeneration den Weg. Transistoren arbeiten ähnlich wie Elektronenröhren, sind aber als Schalter, was den Raumbedarf und die Herstellungskosten betrifft, um Klassen besser. Auch heutige Computer verwenden Transistoren als Schalter – allerdings nicht mehr als einzeln verpackte Bauelemente. Bis zu einer Viertelmillion und mehr Transistoren finden auf einem einzigen fingernagelgroßen Siliziumchip Platz. Sie sind mit bloßem Auge nicht zu erkennen, erfüllen aber die Funktion eines vollwertigen Schalters.

Die Herstellungskosten sanken rapide, weil die vielen tausend Schaltelemente, auf kleinstem Raum gepackt, in einem Herstellungsvorgang produziert werden konnten. Die 1950 noch raumgroßen Rechenanlagen sind heute auf handliche Microcomputer zusammengeschrumpft, die sich fast jeder leisten kann.



Der Nobelpreis wurde 1956 drei Wissenschaftlern der 'Bell-Telefon-Laboratorien' zugesprochen, deren Forschung 1947 zur Erfindung des Transistors geführt hatte. Das Bild zeigt die Wissenschaftler Dr. John Bardeen, Dr. William Shockley und Dr. Walter Brattain (v. l. n. r.)



Associated Press Ltd

Computerkarrieren

Die ständig expandierende Microindustrie hat eine Fülle neuer Berufe und Studiengänge hervorgebracht, die gute Aufstiegsmöglichkeiten in dieser zukunftsträchtigen Branche in Aussicht stellen.

Die ständig steigende Zahl von Heimcomputern in Privathaushalten und Schulen hat viele Programmier-Talente geweckt, die ohne diesen Umstand wohl nie auf die Idee gekommen wären, eine EDV-Laufbahn einzuschlagen. Aber wie so oft zeigt sich auch im EDV-Bereich, daß lückenhaftes Programmier-Wissen, das sich lediglich auf BASIC beschränkt, unzureichend ist.

An professionelle Programmierer werden höhere Anforderungen gestellt, als an Heimcomputer-User. Die Arbeit unterscheidet sich grundsätzlich voneinander.

Für den computerbegeisterten Schulabsolventen scheint ein Informatik-Studium oder der sofortige Eintritt in ein EDV-Unternehmen die richtige Wahl zu sein. Viele Fachhochschulen und Universitäten bieten Ingenieur-Studiengänge an, die zum Einsatz im EDV-Bereich qualifizieren. Absolventen dieser Studiengänge können durchweg mit guten Beschäftigungsaussichten rechnen.

Beschäftigte in der EDV-Industrie sind nur selten von Arbeitslosigkeit bedroht. Wenn überhaupt, trifft es die unteren Ränge der Computer-Hierarchie. Ingenieure, Systemanalytiker und Konstrukteure sind nach wie vor sehr gefragte Leute.

Computern als Schulfach

Auch an Schulen wird verstärkt der Umgang mit Computern angeboten. Bis vor kurzem war dies nur an Fachhochschulen und Universitäten der Fall. Lehrer haben äußerst selten eine EDV-Fachausbildung, und der große Nachholbedarf in diesem Bereich macht das Fach besonders attraktiv.

Durch den Einsatz von Computern sind viele neue Berufe entstanden, die natürlich unterschiedlich hohe Anforderungen an die Beschäftigten stellen. Angelernte Anwender, beispielsweise in der Textverarbeitung oder der Buchhaltung, eignen sich diese Fähigkeiten oft neben der normalen Tätigkeit als Sekretärin oder Buchhalter an. Aber auch andere EDV-Tätigkeiten, wie Dateneingabe (Datatypistin) oder Lochkartenaufbereitung (Locherin) gehören dazu. Diese neuen Tätigkeiten erfordern meist eine abgeschlossene Lehre und die Fähigkeit, klar zu denken. Reine manuelle Fertigkeiten wie das Bedienen einer Tastatur werden normalerweise bei der Arbeit am Gerät ganz einfach erlernt.

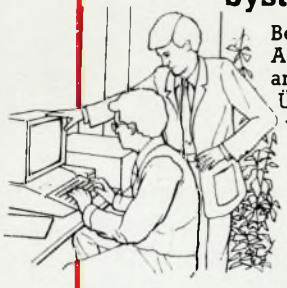
Als nächstes kommt das Bedienungspersonal, das direkten Kontakt zum Computer hat. Und obwohl die in der EDV verwendeten Computer sich in Aussehen und Bedienung erheblich von Heimcomputern unterscheiden, gibt es gemeinsame Grundprinzipien. Vertrautheit mit Heimcomputern ist also durchaus vorteilhaft. Als Bediener begreift man die grundsätzlichen Computerfunktionen sehr schnell und erwirbt so eine gute Ausgangsposition für den Aufstieg zum Programmierer. Man sollte jedoch nicht übersehen, daß der Beruf des Programmierers mit erheblichen körperlichen Belastungen verbunden sein kann, weil viele große EDV-Zentren rund um die Uhr arbeiten.

Die Haupteigenschaften eines angehenden Programmierers sind klares, methodisches Denkvermögen und die Fähigkeit sich auf kleinste Einzelheiten konzentrieren zu können.

Ein großes Team hochspezialisierter Fachkräfte ist erforderlich, um große kommerzielle Computer wie diesen mit hoher Effizienz zu betreiben. Auf Maschinen dieser Größenordnung können Hunderte von Programmen gleichzeitig laufen. Über Telefon, Microwellenverbindung oder über Kommunikationssatelliten können Tausende von Kunden überall in der Welt bedient werden. In einem Raum sind häufig mehrere kommunizierende Großcomputer installiert.

Die Wahl fürs Leben

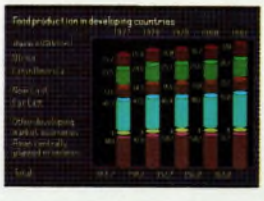
Systemanalytiker

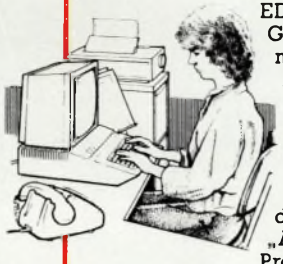


Bevor man eine Aufgabe in Angriff nimmt, ist es angebracht, sich einen Überblick über Ziele und vorhandene Lösungsmöglichkeiten zu verschaffen. Der **Systemanalytiker** ermittelt durch Befragung die Anforderungen des Computer-Anwenders und definiert das Problem. Er entwickelt daraufhin eine Strategie, die mit den vorhandenen Mitteln eine Problemlösung erlaubt. Um zu praktisch durchführbaren Vorschlägen zu gelangen, braucht er die Fähigkeit zu logischem Denken, gute Kommunikationsfähigkeit und Kreativität. Häufig ist er auch als „Verkäufer" der EDV-Abteilung tätig.

Programmierer

Der **Programmierer** beginnt mit der vom Systemanalytiker entwickelten Lösungsstrategie. Sie bildet die Grundlage für die Planung seines taktischen Vorgehens. Hierzu gehört das Aufbrechen des Gesamtproblems in kleine, gut zu handhabende Teilprobleme und deren Umsetzung in einen vom Computer lesbaren Code. **Anwendungsprogrammierer** schreiben Programme für begrenzte Anwenderprobleme. **Systemprogrammierer** haben dagegen mehr mit der Funktion der gesamten EDV-Anlage zu tun. Auch innerhalb einer Projektgruppe tendieren Anwendungsprogrammierer eher dazu, allein zu arbeiten. Dies mag an ihrer Fähigkeit liegen, sich besonders auf aktuelle Teilprobleme konzentrieren zu können. Systemprogrammierer brauchen diese Fähigkeit auch, dürfen sich aber nicht so leicht aus der Ruhe bringen lassen.





Bediener kleinerer EDV-Anlagen haben häufig Gelegenheit, Programmierern oder Ingenieuren bei der Fehlersuche zu helfen oder die gerade entwickelten Programme zu testen. Von großer Wichtigkeit ist die genaue Kenntnis des Programmablaufs. „Anwenderfreundliche" Programme erleichtern die Arbeit des Bedieners.

EDV-Abteilungen sind genau wie alle anderen Bereiche eines Unternehmens hierarchisch organisiert. An der Spitze steht der Abteilungs- oder Bereichsleiter. Er ist für alle Aufgaben verantwortlich, die unter „Informations-Verarbeitung" zusammengefaßt werden.

Computer-Fachleute sind in erster Linie „Techniker" und erwerben die Management-Fähigkeiten während ihres beruflichen Aufstiegs im Unternehmen. Drei Hauptbereiche für Spezialisierung sind offensichtlich: Bedienung, Programmierung und Systemanalyse. Zwischen diesen Bereichen gibt es keine harte Abgrenzung, so daß während des Aufstiegs von einem in den anderen Bereich gewechselt werden kann.

Was für andere Arbeitsgebiete gilt, trifft auch für die EDV zu: Der Einstieg sollte mit möglichst hoher Qualifikation erfolgen. Obwohl es anfangs so aussieht, als ob es keinen Unterschied macht, zeigt sich später jedoch, daß der weniger Qualifizierte schnell auf unüberwindliche Hürden stößt. Deshalb werden in der Bundesrepublik eine Vielzahl von Fernlehrgängen staatlich anerkannter Institute angeboten, die eine Aus- und Weiterbildung im EDV-Bereich ermöglichen. Die Zeugnisse genießen auch in der Industrie Anerkennung und ermöglichen dem angehenden Programmierer oder Systemanalytiker einen guten Start.

Bedienungspersonal

Die größte körperliche Belastung bringt die Bedienung eines Computers mit sich, der von vielen Anwendern in ebensovielen Programmiersprachen benutzt wird. Der Bediener muß nicht nur Schichtarbeit leisten und Wechselplatten, Magnetbandspulen oder Pakete voll Tabellierpapier transportieren, er muß sich auch mit dem Betriebssystem des Computers und den laufenden Programmen gut auskennen.

Ein **erfahrener Bediener** wird in eigener Verantwortung über den Zugriff zum Computer entscheiden dürfen und hat so auch Einfluß auf viele andere Abteilungen eines Unternehmens.

An das **Dateneingabe-Personal** werden geringere Anforderungen gestellt. Fähigkeiten, die hier gebraucht werden, sind vergleichbar mit denen, die man in Schreibbüros verlangt: Es muß schnell und genau gearbeitet werden. Die Arbeit kann hier durch häufige Wiederholungen schnell ermüdend wirken.

Entwicklungsingenieur

Die Zeit, in der Computer ihre eigenen Nachfolger konstruieren, scheint nicht mehr fern zu sein. Innovative Ideen hat jedoch nach wie vor der **Entwicklungsingenieur** zu liefern. Er ist Techniker und Wissenschaftler zugleich. Neuentwicklung und ständige Verbesserung der Leistungsfähigkeit vorhandener Geräte durch konsequente Nutzung neuer Entwicklungen sind seine Hauptaufgaben.

Tony Sleep

Service-Ingenieur

Wenn beim Computer eine Störung auftritt, muß der **Service-Ingenieur** die Panne beheben. Die Fähigkeit moderner Computer, eigene Fehler einzukreisen und auf das betroffene, auswechselbare Modul zu reduzieren, erleichtert zwar die Arbeit des Service-Ingenieurs, kann aber fundierte Kenntnisse der digitalen Elektronik nicht ersetzen. Für dieses Arbeitsgebiet ist die Ausbildung als Techniker oder Fachhochschul-Ingenieur eine gute Voraussetzung.

Sicher erfordert der Beruf des Programmierers besonderes Talent, und obwohl ein höherer Schulabschluß die Mindestvoraussetzung ist, zählt mehr die natürliche Begabung, logisch zu denken. Möglichkeiten für den Einstieg als Programmierer in die EDV ohne besondere formale Qualifikation bestehen hauptsächlich in den USA und England, weniger hierzulande. Trotzdem sollte die Programmierfreudigkeit vieler Mädchen und Jungen bei uns die Eltern hoffnungsvoll stimmen.

Profitable Spielerei

Wie schon erwähnt, machen BASIC-Erfahrungen allein noch keinen Programmierer und öffnen nicht unbedingt die Türen zur EDV-Industrie. Trotzdem sind viele Jugendliche in der Lage, aus ihren BASIC-Kenntnissen ein profitables Geschäft zu machen. Viele schreiben ihre Computerspiele in BASIC, die großes Interesse bei anderen Jugendlichen wecken, und auch kommerzielle Software-Anbieter wollen Spiele in ihr Programm aufnehmen, die das jugendliche Gemüt so gezielt ansprechen. Einige dieser „Programmzauberer", die in Zeitungen und Zeitschriften als Großverdiener genannt werden, programmieren in BASIC und verfügen selten über tiefschürfende Computerkenntnisse. Andere sind bewundernswerte Talente. Sie programmieren in Assemblersprache (eine Programmiersprache, direkt über der Maschinensprache angesiedelt, die den Microprozessor-Maschinencode sehr effizient nutzt) und haben ausgezeichnete Berufschancen.

In der EDV-Industrie gibt es zwei Gruppen von Programmierern: Anwendungs- und Systemprogrammierer. Anwendungsprogrammierer schreiben Programme für die Ausführung eng begrenzter Aufgaben. Systemprogrammierer halten das Computersystem in Ordnung. Ein Beispiel ist die Fehlersuche. Anwendungsprogrammierer haben auch Kontakt mit Leuten außerhalb des EDV-Raumes, so z. B. mit Kunden, und Teamarbeit ist bei der Entwicklung maßgeschneiderter Programme selbstverständlich. Systemprogrammierer sind dagegen ausgesprochene Spezialisten. Ihr enger Kontakt mit der „Intelligenz" des Computers fördert die Neigung zum Alleinarbeiten.

Hier nun zieht die EDV-Industrie eine Trennungslinie, die nur die fähigsten Programmierer und hochqualifizierte Hochschulabsolventen überschreiten. Jenseits dieser Linie liegt nämlich das Reich der Systemanalytiker und Computer-Konstrukteure.

Systemanalytiker finden heraus, wie mit einem Computer ein Problem gelöst werden kann. So hat beispielsweise ein Ölkonzern neue Vorkommen unter dem Meeresboden entdeckt. Beim Feststellen der Ergiebigkeit des Vorkommens wurde auch ermittelt, daß die Ölqualität sehr schwankt. Die Konzernleitung muß nun entscheiden, ob die erforderlichen Milliarden investiert werden sollen. Auch muß die Situation des internationalen Ölmarktes für den Ausbeutungszeitraum des Vorkommens (etwa 20 Jahre) berücksichtigt werden. Gleichzeitig ist festzulegen, an welcher Stelle des Ölfeldes die erste Bohrung niedergebracht werden soll. Weil die Investition so riesig ist, übergibt die Konzernleitung das Problem den EDV-Fachleuten. Die Systemanalytiker ziehen Ökonomen, Marketing-Experten, Geologen und andere Spezialisten zu Rate und sind in der Lage, ein Computer-Modell des Erdölfeldes zu entwerfen.

Die Führungskräfte des Konzerns können nun am Modell durchspielen, wie sich verschiedene Entscheidungen über Preis, Marktstrategien und Aufbereitungsverfahren auf die gesamte Leistungsfähigkeit des Projektes auswirken. Das „Spiel" heißt „Was geschieht, wenn?" und liefert alle Informationen für die endgültige Entscheidung.

Es gibt mehrere wichtige Funktionen in der EDV-Industrie, die des Systemanalytikers genießt jedoch das höchste Ansehen. Eine Ausnahme macht vielleicht der Hardware-Konstrukteur. Für Elektronik-Ingenieure existieren viele Beschäftigungsmöglichkeiten; vom Service-Zentrum bis zur Forschungsabteilung. Arbeitsplätze in den Entwicklungsabteilungen und der reinen Forschung werden mit den bestqualifizierten Fachleuten besetzt. Viele Analytiker sowie Konstrukteure von Soft- und Hardware steigen ins Management auf oder machen sich als beratende Spezialisten selbständig. Obwohl sich die Berufsbezeichnung ändert, die oft nur gestiegene Entscheidungsgewalt anzeigt, bleibt das Tätigkeitsfeld des Computer-Experten vielfach gleich.

An geschultem Computer-Personal besteht nach wie vor großer Bedarf. So rechnet man in England mit mindestens 20 000 nicht zu besetzender Stellen, obwohl gleichzeitig eine große Zahl von graduierten Abgängern der Universitäten arbeitslos ist. Die sich hier zeigende Fehlanpassung von Ausbildung und Arbeitsplatzangebot ist Ursache für ernste Bedenken sowohl in der Wirtschaft als auch im Bildungssystem. Umschulungsprogramme und wesentlich erweiterte Möglichkeiten an den öffentlichen Schulen sollen hier Abhilfe schaffen.

In vielen europäischen Ländern und besonders in Großbritannien hat sich die Überzeugung durchgesetzt, daß zumindest einem Teil der Kurzzeit-Arbeitslosigkeit durch Mikroelektronik begegnet werden kann. Man versucht durch besondere „on the job"-Trainingsprogramme arbeitslosen Schulabgängern praktische Erfahrungen zu vermitteln. Zu Beginn des Programms wurden 4500 Trainingsplätze von den dortigen 'Information Technology Centres' angeboten. Hier können sich junge Schulabgänger Computerwissen aneignen und erhalten zudem eine Bezahlung.

David Simmonds ist 17 und hat im vergangenen Jahr während seiner Sommerferien gut 10 000 englische Pfund (rund 40 000 DM) verdient. Er ist ein „Wunderkind der Branche" und schreibt Programme für Commodore. Im Unterschied zu vielen gleichaltrigen Teenagern schreibt David seriöse Software für kommerzielle Anwendungen. Er ist zuversichtlich, nach Abschluß seines Studiums eine lukrative Nische im EDV-Markt zu finden. Das Ganze begann mit Computer-Spielen, als sein Vater einen Computer mit nach Hause brachte. David verlor das Interesse am Spielen, als er entdeckte, wie man den Computer programmiert.

Eugene Evans ist ebenfalls 17, und seine Einnahmen sollen bei 40 000 englischen Pfund (ca. 160 000 DM) pro Jahr liegen! Eugene ist einer der vielen „Programmzauberer" des Computer-Booms. Der talentierte Jungprogrammierer erstellt für seine Arbeitgeber interessante Computerspiele, die bei den Teenagern, den Hauptabnehmern dieser Ware, hervorragend ankommen. Die hohen Einkünfte bestehen aus Lizenzgebühren aus dem Verkauf der Spiele (ähnlich wie bei Buchautoren).

Miniaturtechnik auf Sand gebaut

Silizium ist der Rohstoff, aus dem Chips hergestellt werden. Ein aufwendiger Fertigungsprozeß geht dem Endprodukt voraus.

Silizium ist, neben Sauerstoff, das auf der Erde am häufigsten vorkommende Element. Aus ihm bestehen alle Sandkörnchen der Welt. Die Bedeutung für die microelektronische Industrie liegt in der physikalischen Struktur des Siliziums. In seiner reinsten Form ist es ein schlechter Elektrizitätsleiter. Wenn man es aber mit einer genau dosierten Menge bestimmter Fremdkörper verunreinigt, wird es zu einem Halbleiter.

Die elektrische Leitfähigkeit eines Materials wird durch Anzahl und Art der Bindung der Elektronen bestimmt, die die Atomkerne umgeben. In Metallen wird Strom durch fast ungebundene Elektronen transportiert. Diese können sich innerhalb des Metalls frei bewegen und übertragen ihre Bindung und elektrische Ladung von einem Atom zum nächsten. Ein Isolator kann keinen Strom leiten, weil er keine freien Elektronen besitzt.

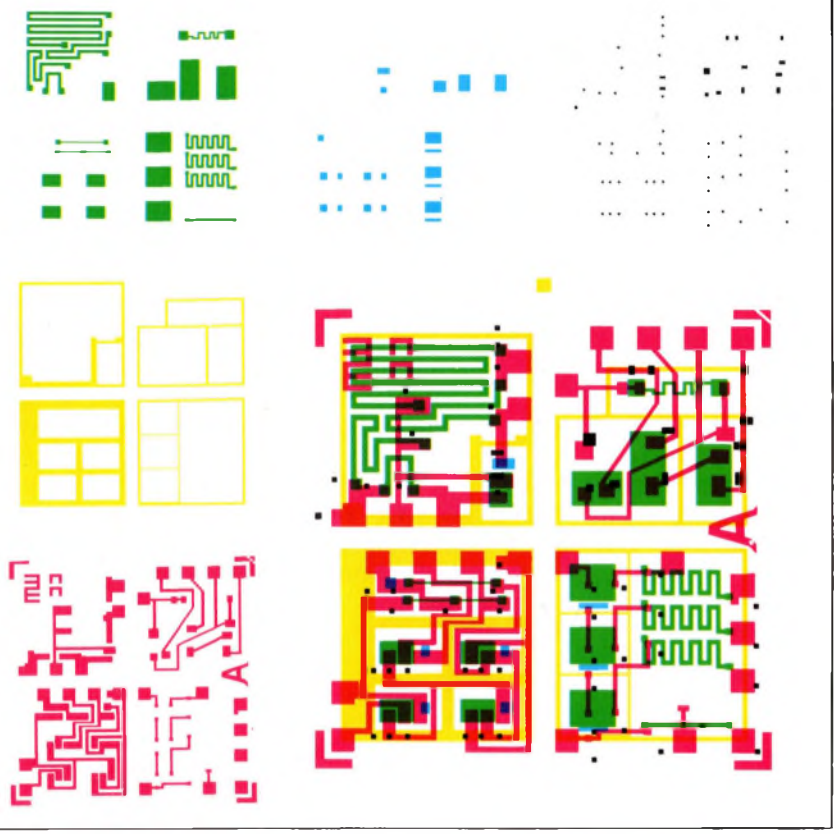
Die Herstellung reinen Siliziums ist ein relativ einfacher Prozeß. Zuerst wird das Oxyd chemisch bis zu 99,99prozentiger Reinheit veredelt. Dann wird es in einem Tiegel bis zum Schmelzpunkt (1410 °C) erhitzt. Dies geschieht in einer Edelgasatmosphäre, um störende Elemente abzuhalten. Für die Dotierung, die kontrollierte Verunreinigung des Siliziums, verwendet man Phosphor und Bor. Dabei bewirkt Phosphor eine n-Dotierung (negative Ladung). Bor ruft eine p-Dotierung hervor. Man erhält große Kristalle, indem man einen perfekten Einkristall in die Schmelze taucht und diesen langsam drehend wieder hochzieht. Auf diesem Weg erhält man Kristalle mit einem Durchmesser von sieben bis zehn und einer

Ferrari

Das Foto zeigt einen Silizium-Brennofen. Nachdem das Silizium gereinigt, zersägt, belichtet und entwickelt wurde, erhält es eine Oberfläche aus Siliziumdioxyd. Dazu werden die Wafer auf 1050 °C erhitzt, und reiner Sauerstoff wird darübergeleitet. Während sie in einem ‚Boot' aus geschmolzenem Quarz im Ofen liegen, bildet sich eine Isolierschicht aus Siliziumdioxyd SiO_2. Diese Schicht wird dann beim nächsten Entwickeln teilweise entfernt, und dieser Zyklus wiederholt sich für jede weitere Schicht.

Die Summe der Teile

Jedes Foto wird in vier Farben und Schwarz aufgespalten. Diese werden dann einzeln übereinander gedruckt, um den hier sichtbaren Effekt zu erzielen. Die Chipherstellung benutzt Schichten aus dotiertem Silizium und anderen Materialien. Das Drucken ist wesentlich einfacher als das Beschichten/Belichten/Entwickeln der Chips. Wie man auf dem Bild sieht, werden die einzelnen Masken nacheinander kombiniert, um den Schaltkreis aufzubauen.



Länge von gut 60 Zentimetern. Diese werden dann auf einen Durchmesser von 76 oder 100 mm abgeschliffen. Der Kristall wird in Scheiben zerschnitten, beidseitig abgeschliffen, jedoch nur von einer Seite poliert. Der fertige Wafer, eine Siliziumscheibe, auf der integrierte Schaltkreise aufgetragen werden, ist dann nur noch einen halben Millimeter dick.

Wenn aber der Prozeß so einfach und das Rohmaterial so billig ist, warum ist das reine Silizium so teuer, bis zu 40 DM pro Scheibe?

Die Antwort liegt in der absoluten Reinheit des Materials. Man muß genau darauf achten, daß nicht ein einziges Fremdatom ins Silizium gelangt. Die Luftreinheitswerte in Siliziumfabriken sind beachtlich, sie betragen weniger als 3000 Partikel pro Kubikmeter. Damit ist die Luft dort mehr als 100mal sauberer als in modernsten Krankenhäusern.

Fotolithografie

Die Herstellung eines integrierten Schaltkreises, eines ICs, benötigt Microgravierungen auf der Chipoberfläche. In der Massenproduktion wird dies durch einen Prozeß erreicht, der als Fotolithografie bekannt ist.

Jede Schicht des Schaltkreises wird dabei als separate Einheit behandelt. Die Originalvorlagen werden mit Hilfe von Computern entwickelt, abfotografiert und zuletzt auf die geforderte Größe verkleinert. Die Maske selbst wird produziert, indem man das Foto vervielfältigt und gitterartig zusammensetzt, um den gesamten Wafer zu bedecken.

Zuerst wird der Wafer in reinem Sauerstoff auf 1050 °C erhitzt, um eine Isolationsschicht aus Siliziumdioxyd herzustellen. Diese Schicht wird später selektiv entfernt, und man erhält ‚Fenster' zum reinen Silizium darunter.

Die oxidierte Oberfläche des Wafers wird zuerst mit einem lichtempfindlichen Film beschichtet, dessen Löslichkeit sich unter UV-Licht extrem erhöht. Dann wird die Maske der ersten Schicht des Schaltkreises auf den Wafer gelegt, die freigebliebene Oberfläche wird UV-Licht ausgesetzt und in einer Flüssigkeit entwickelt, die den gesamten Film vollständig entfernt.

Eine ähnliche Methode wird bei Kupferstichen angewendet. Die Kupferoberfläche wird mit Wachs beschichtet, der Entwurf ausgekratzt und in eine schwache Säure getaucht. Wenn das Wachs entfernt ist, bleibt der Entwurf sichtbar. Unter der Wachsschicht fand keine chemische Reaktion statt.

Nachdem die erste Stufe beendet ist, wiederholt sich der Prozeß mit neuen Masken und Chemikalien, bis der Chip komplett ist. Zum Schluß wird der gesamte Wafer nochmals mit einer Isolation aus Siliziumdioxyd beschichtet. Dieser Vorgang kann mehr als zehn Wiederholungen (Beschichten/Belichten/Entwickeln) erfordern. In jeder Stufe können viele Fehler

auftreten, und die Ausschußrate in der Prüfphase ist hoch. Je komplexer der Schaltkreis und je dichter der Wafer gepackt ist, desto höher ist der Ausschuß.

In der ersten Teststufe kommt der gesamte Wafer mit seinen Hunderten von Schaltkreisen in ein computerkontrolliertes Gerät, das alle Chips testet. Dazu sind mehrere Chips auf dem Wafer nur für Testzwecke reserviert. Die Prägeeinheit markiert nicht nur jeden Ausfall mit einem Tintenklecks, sondern liefert auch Informationen über Ausfallraten und Positionen der Fehler bei Chips und Wafer. Danach werden die Wafer in einzelne Chips zerlegt und die markierten aussortiert. Die verbleibenden werden auf Stempeln befestigt.

Der Chip wird mit sehr feinen Drähten an den Stempel angeschlossen, wiederum vom Computer kontrolliert. Das Ganze wird in ein Plastik- oder Keramikgehäuse eingegossen, aus dem die Anschlußbeine (Pins) herausragen, bevor die letzte Prüfung ansteht. Gegenwärtig liegt die Miniaturisierungsgrenze bei der Wellenlänge des UV-Lichts, also bei etwa einem 20millionsten Teil eines Meters. Neuerdings setzt man auf die Verwendung von Röntgenlicht, das voraussichtlich eine weitere Verdichtung um den Faktor 50 ergeben wird.

Alle Aspekte der Chipherstellung, vom Design über den Prototyp bis hin zur Massenproduktion und Endprüfung, sind so komplex, daß sie ohne Unterstützung derjenigen Einheiten, die gerade hergestellt werden, gar nicht realisierbar wären – ein echtes Paradoxon.

Produktionshilfe vom Computer

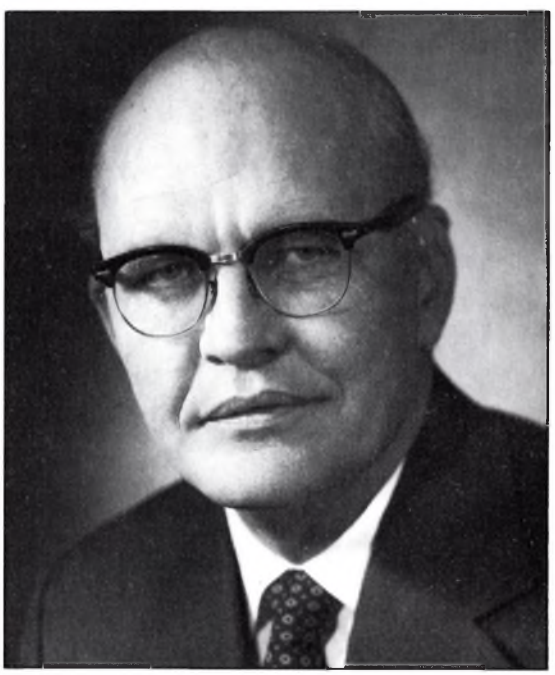
Die Verwendung von Computern, zum Beispiel in der Designphase, ermöglicht es, einzelne Sektionen einmalig zu definieren und sie wieder aus dem Speicher abzurufen, wenn sie benötigt werden. Nehmen wir beispielsweise einen Schreib/Lesespeicher (RAM). Jedes Bit benötigt eine Zelle mit je einem Transistor. Um also 2KByte zu speichern, muß der Chip 16 384 identische Zellen enthalten. Der Designer braucht also nur eine Zelle zu definieren und den Designcomputer anzuweisen, diese 16 384 mal zu multiplizieren. Die Nutzung von Computern beim Design ist nicht nur auf diese praktische Anwendung beschränkt. Die Konstrukteure arbeiten nicht an Zeichenplatten, sondern an Computerterminals, und zeichnen mit Lightpens direkt auf dem Bildschirm. Der fertige Entwurf wird dann vom Rechner auf einem Mehrfarbplotter ausgegeben. Nicht nur das Design der Masken wird mit Hilfe des Computers verbessert. Spezielle Programme, die auf ein bestimmtes Problem abgestimmt sind, werden auch für die Schaltkreisentwicklung eingesetzt. Dies ermöglicht dem Konstrukteur, verschiedene Lösungsmöglichkeiten auszuprobieren, bevor er sich für die geeignetste Variante entschließt. Dadurch lassen sich kostenintensive Tests von unausgereiften Prototypen vermeiden.

Wenn wir von der microelektronischen Revolution sprechen, sind auch die enorme Kostensenkung und die Miniaturisierung der Computer gemeint. Deshalb sollten wir uns die Nachteile eines alten Röhrenrechners ins Gedächtnis rufen:

Enormer Platzbedarf
Geringe Arbeitsgeschwindigkeit
Großer Energiebedarf
Beschränkter Speicherplatz und Befehlssatz
Hohe Kosten

All diese Bereiche waren unmittelbar von der Entdeckung des Transistors betroffen, wobei aber die Computerherstellung immer noch arbeitsintensiv und teuer blieb, weil die Bauteile auf gedruckten Leiterplatten zusammengefügt werden mußten.

Erst die Entwicklung des integrierten Schaltkreises, und vor allem des Microprozessors, ermöglichten der Industrie die volle Nutzung aller Vorteile computerkontrollierter Prozesse und Herstellungsverfahren.



Texas Instruments

Jack Kilby wird die Entdeckung des Integrierten Schaltkreises zuerkannt, den er 1958 bei Texas Instruments entwickelte. Er konstruierte ein Modul von 1,3 cm × 0,6 cm Größe, das mehrere Transistoren enthielt. Moderne Chips vereinigen Hunderttausende Komponenten auf dem gleichen Platz.

Die Statistiken sind teilweise atemberaubend. 1959 gab es nur Bauteile wie Transistoren und Dioden. Schon 1978 konnten LSI-Schaltungen (Large Scale Integrated circuit) bis zu einer viertel Million Teile auf einem einzigen Chip enthalten. Von 1973 bis 1983 sanken die Kosten je Bit-Speicherplatz um den Faktor 20, und die Nutzung elektronischer Geräte jeder Art stieg weltweit um den Faktor 1000. Diese Trends werden sich in der näheren Zukunft wohl fortsetzen. Es wird geschätzt, daß die Anzahl der genutzten Groß- und Kleincomputer während der nächsten drei Jahre um das Hundertfache ansteigen wird – besonders deshalb, weil die Geräte günstiger werden.

John von Neumann

Der brillante Mathematiker gehört zu den Wegbereitern der modernen Computertechnologie. Die von ihm entwickelte Monte Carlo-Methode ermöglicht die Ermittlung von Zufallszahlen per Rechner.

John von Neumann mit seiner zweiten Frau Klara, einer ausgezeichneten Programmiererin.

Nur ein Ungar kann eine Drehtür hinter Ihnen betreten und vor Ihnen wieder verlassen." So beschrieb John von Neumann einmal den Ehrgeiz seiner Landsleute; er selbst war keine Ausnahme. Seine Zielstrebigkeit und seine genialen mathematischen Fähigkeiten verhalfen ihm zu einem der höchsten wissenschaftlichen Posten der Vereinigten Staaten.

Neumann war das Kind einer wohlhabenden jüdischen Familie im österreichisch-ungarischen Reich. Sein mathematisches Talent wurde in frühester Jugend entdeckt. Im Alter von 25 Jahren besaß er bereits einen Doktortitel und zwei akademische Grade.

Er wußte immer seinen Vorteil zu nutzen. So nahm er nach dem Zusammenbruch des österreichischen-ungarischen Reiches den Titel ‚von' an und arbeitete im besiegten Deutschland. Zur selben Zeit baute er Kontakte in Amerika auf. Neumann verbrachte die Winter an der Princeton University in New Jersey und verwaltete im Sommer das Vermögen seines Vaters in Europa. Als der Zweite Weltkrieg ausbrach, hatte er sich bereits in Amerika etabliert. Neumann machte sich einen Namen in der Mathematik durch die Neuordnung der Gruppentheorien, die Bertrand Russell mit seinen logischen Paradoxa in Frage gestellt hatte. Er war fasziniert von der Quantenphysik und den verschiedenen Spieltheorien in der Wirtschaftsmathematik.

Als der Krieg Amerika erreichte, wurde er an das sogenannte Manhattan-Projekt in Los Alamos berufen, wo er begeistert die Atombombe mitentwickelte.

Neumann war noch am Manhattan-Projekt beschäftigt, als er erfuhr, daß man einen elektronischen Computer entwickeln wollte. Daraufhin bot er sich für das ENIAC-Vorhaben an. Die Entwicklung wurde bis dahin von Ingenieuren geleitet. Zunächst einziger Mathematiker in dem Projekt, ging er das Problem von einer völlig anderen Seite an. Schließlich entwarf er einen Bericht, der zur Grundlage für die modernen Computer-Konstruktionen unserer Zeit wurde.

Nach dem Krieg widmete er seine Arbeit zunehmend der Verteidigungspolitik der Vereinigten Staaten. Allerdings trieb er weiterhin mathematische Forschungen und entwarf den ersten Computer, genannt JOHNIAC, für die Princeton University. Zur Einweihungsfeier des Gerätes wurde ein aus Eis gefertigtes Modell des JOHNIAC vorgestellt.

Etwa im Alter von 50 Jahren erkrankte Neumann an Krebs. Später mußte er infolgedessen in einem Rollstuhl sitzen. Obwohl er als Agnostiker sein Leben lang nur sinnliche Wahrnehmungen gelten ließ, setzte er sich kurz vor seinem Tod immer stärker mit religiösen Fragen auseinander. Über seine letzten Jahre sagte ein Freund: „Ich glaube, von Neumann litt mehr als irgendein anderer Mensch, den ich gesehen habe, unter seiner mehr und mehr nachlassenden Geisteskraft."

Neumann hinterließ eine Reihe bewältigter

1903
Neumann wird am 28. Dezember in Budapest geboren

1921
Erste mathematische Veröffentlichung

1933
Professur an der Princeton University, New Jersey

1943
Ernennung zum Berater des Manhattan-Projekts in Los Alamos, New Mexico

1947
Bericht über ENIAC und Konstruktionsentwurf für einen neuen Computer, genannt EDVAC (Elektronischer Diskreter Variabler Computer), der unter Verwendung der ENIAC-Pläne gebaut werden sollte

1951
Einsatz des selbstentwickelten Computers JOHNIAC an der Princeton University

1951–1953
Präsident der amerikanischen Gesellschaft für Mathematik, Arbeit an der Automatentheorie

1955
Ernennung zum Atomenergie-Beauftragten durch Präsident Eisenhower

1957
Stirbt am 8. Februar an Krebs. Zur Ehrung des Mathematikers wird eine ganze Ausgabe des ‚Bulletin of the American Mathematical Society' Neumanns Leben und Werk gewidmet

mathematischer Probleme – so beispielsweise die von ihm entwickelte Monte Carlo-Methode, durch die Gleichungen unter Verwendung von Zufallszahlen gelöst werden können. Voraussetzung für dieses Verfahren war die Ermittlung von Zufalls- (randomisierten) Zahlen. Und genau dieses Problem galt es zu lösen. Computer arbeiten vollständig rational, die zufällige Auswahl von Zahlen aber ist irrational. Sie können bis heute keine echten Zufallszahlen erzeugen, denn eine solche Zahl ist nicht voraussagbar. Ein Beispiel dafür sind die Roulette-Ergebnisse – daher hat die Methode auch ihren Namen. Ein Computer befolgt nur Befehle und benötigt für jede Aktion bestimmte Voraussetzungen. Daher ist jede Zahl, die er erzeugt, die Folge einer Serie von Befehlen. Gleichgültig wie kompliziert diese Anweisungen sind, jede Zahl ist voraussagbar. Um sie zu finden, muß man nur dem Befehl folgen, den der Computer ausführt. Dadurch ergibt sich jedoch nicht wirklich eine rein zufällige Zahl.

Neumann war es, der auf die Idee der Pseudo-Zufallszahlen kam – Zahlen, die mathematisch ermittelt werden. Nach seiner Methode nimmt man irgendeine vierstellige Zahl, zum Beispiel 4321, und quadriert sie. Aus dem achtstelligen Ergebnis (18671041) werden die mittleren vier Ziffern (6710) als Zufallszahl verwendet. Mit dieser wird dann derselbe Prozeß wiederholt. 6710 quadriert ergibt 45024100. Man erhält somit als nächste randomisierte Zahl 0241 und so weiter. Dieser Quadrierungsprozeß kann zwar beliebig weitergeführt werden, aber die mögliche Anzahl der Zufallszahlen ist begrenzt (9999). Deshalb wird sich die Reihenfolge der Zahlen irgendwann wiederholen. Moderne Computer benutzen Methoden, mit denen wesentlich mehr Pseudo-Zufallszahlen erzeugt werden können.

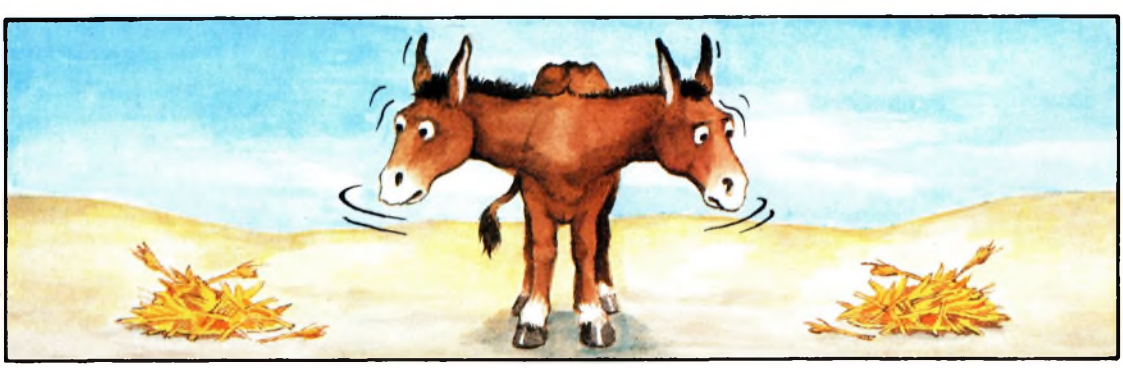
Zufallszahlen wurden erstmals von Telefontechnikern benutzt. Dadurch konnte die unterschiedliche Häufigkeit von Vermittlungen des Fernmeldeamtes simuliert werden. Auf diese Weise ließen sich mögliche Störfaktoren frühzeitig erkennen. Heutzutage gibt es vielfältige Anwendungen der Zufallszahlen, von Computerspielen über die Simulation von Schwankungen bis hin zur Lösung komplizierter mathematischer Funktionen.

Ein weiteres Beispiel ist die Berechnung der Fläche eines Landes, von dem nur eine Umriß-Karte vorliegt. Die Abbildung unten zeigt eine solche Karte von Großbritannien. Die Gesamtgröße der Fläche von Wasser und Land ist bekannt: 577 500 km^2. Mittels Zufallszahlen werden 40 Punkte auf der Karte ‚verstreut'. Die Punkte sind aufgrund der randomisierten Zahlen gleichmäßig über die ganze Karte verteilt. Die Anzahl der Punkte, die auf die Landmasse

Kevin Jones

Auf dieser Karte von Großbritannien wurden 40 Punkte mit Hilfe randomisierter Zahlen ‚verstreut'. 24 Punkte fallen ‚ins Meer' und 16 treffen das Land. Aus diesen Angaben kann die Fläche des Landes berechnet werden, wenn die Gesamtfläche der Karte bekannt ist.

fällt, ist folglich proportional zu der Fläche des Landes. 24 Punkte entfallen auf die Meeres- und 16 auf die Landfläche. Die Fläche läßt sich folgendermaßen berechnen: (577 500 : 40) x 16 = 231 000 km^2. Bei einer größeren Anzahl von Punkten ergibt sich eine bessere Annäherung an den tatsächlichen Wert von 229 523 km^2.

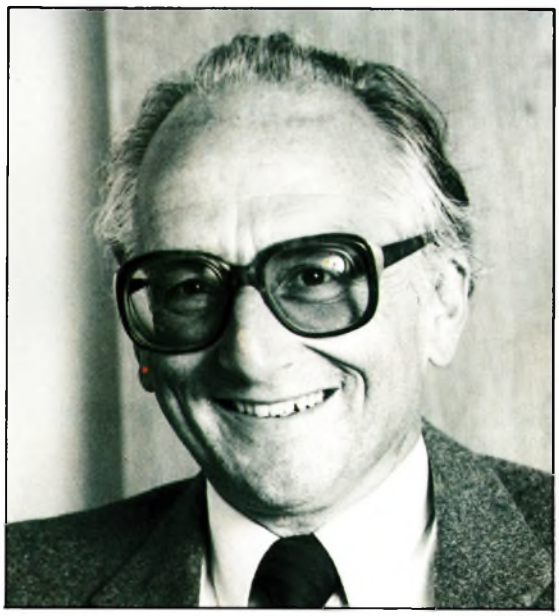


David Higham

Was ist rationales Denken? Ein eigentlich dafür untypisches Tier soll dies verdeutlichen. Angenommen, man stellt einen vollkommen rational (vernünftig) denkenden Esel in die Mitte zweier gleich großer Heuhaufen. Der Esel kann sich nicht den größeren Haufen aussuchen, weil beide gleich groß sind. Er kann sich auch nicht dem näheren zuwenden, weil er genau in der Mitte steht. Weil es keinen logischen Grund für den Esel gibt, einen der Heuhaufen dem anderen vorzuziehen, wird er überhaupt keinen fressen.

Chuck Peddle

Der Computerpionier entwickelte den Microprozessor 6502 und einen der ersten Personalcomputer.



Chuck Peddle machte sich daran, einen Personalcomputer zu entwickeln. Der sollte ein in sich abgeschlossenes Gerät sein, das nach dem Einstecken in die Steckdose sofort einsatzbereit ist. Das Ergebnis war der PET. Fast zur selben Zeit erschien der Apple II. PET zeichnete sich aus durch: eingebauten Monitor, integriertes Cassettengerät und den bekannten BASIC-Interpreter von Microsoft. Der PET hat viele Änderungen überstanden und ist weiterhin populär. Dazu hat besonders die Umstellung auf eine vollwertige und somit verständliche Schreibmaschinentastatur beigetragen.

Er ist eine Generation älter als die Wunderkinder unter den Computerkonstrukteuren, Stephen Wozniak und Steve Jobs. 1973 ging Chuck Peddle zu Motorola, um unter anderem bei der Entwicklung des 6800 Microprozessors mitzuarbeiten.

Der 6800, einer der ersten Microprozessoren auf dem Markt, war mit einem Stückpreis von 200 Dollar entsprechend teuer. Chuck Peddle fand, daß dieser Preis eine breite Markteinführung verhindern würde, und verließ Motorola, um bei der Firma MOS Technology neu zu beginnen.

Was er in diesem relativ kleinen Unternehmen entwickelte, sollte bald der erfolgreichste Microprozessor des ersten Microcomputer-Jahrzehnts werden – die 6502 MPU. Niemand konnte erahnen, daß dies der Grundstein einer ganzen Industrie war, die eine soziale Umwälzung von enormen Ausmaßen in Gang setzte.

Einer der schon sehr früh die weitreichenden Auswirkungen von Microprozessoren und speziell des 6502 von MOS Technology erkannte, war Jack Tramiel, Ex-Präsident von Commodore. Bis dahin waren Commodores Umsätze mit einer Reihe von Büroprodukten und den verschiedensten Taschenrechnern recht bescheiden gewesen.

Es ist nicht schwer zu verstehen, daß Tramiel, Hauptkunde von 4-Funktionen-Chips für Taschenrechner von MOS Technology, diese Firma eines Tages aufkaufte, obwohl Commodore selbst alle Mühe hatte, den Kopf über Wasser zu behalten. Wichtigster Bestandteil des Kaufvertrages aber war für Tramiel, neben dem 6502, der Entwicklungsingenieur Chuck Peddle.

Chuck Peddle hatte sicher auch mit seinem Konzept eines Personalcomputers Eindruck gemacht. Ähnliche Ideen verfolgten, unabhängig von Peddle, die Apple-Gründer Wozniak und Jobs. Chuck Peddle war so sehr um seine Idee besorgt, daß er sich mit Bill Gates, dem Gründer von Microsoft und Vater des berühmten BASIC-Interpreters, zusammentat, um Apple aufzukaufen. Dies geschah fast zur gleichen Zeit, als Commodore MOS Technology erwarb. Wozniak und Jobs forderten damals 150 000 Dollar für Apple, aber Peddle und Gates konnten nur zwei Drittel davon aufbringen.

So blieb Chuck Peddle bei Commodore und übernahm 1977 die Entwicklung des PET (Personal Electronic Transactor). Zur selben Zeit bauten Wozniak und Jobs gerade den Apple II. Der PET unterschied sich davon durch eingebauten Monitor, integriertes Cassettengerät sowie eine Tastatur, deren Qualität allerdings mehr mit einem Taschenrechner als einer Schreibmaschine vergleichbar war. Ungeachtet dieses Nachteils hatte Commodore im Handumdrehen 1000 PETs zum Preis von etwa 1500 Dollar das Stück verkauft. Damit war die erste Generation von Microcomputern geboren, die speziell für den privaten Hausgebrauch entwickelt wurde.

Drei Jahre später entwickelte Chuck Peddle ein neues Talent – das des Firmenchefs. Zusammen mit Chris Fish, einem ehemaligen Finanzfachmann von Commodore, gründete er Sirius Systems Technology als Tochterunternehmen der Walter Kidde Corporation.

Die Entwicklung im Bereich der Personalcomputer konzentrierte sich damals auf die 16-Bit-Chips wie den 8088 von Intel. Auch bei IBM bastelte man fieberhaft am PC. Sirius brachte den Sirius I jedoch einige Wochen früher auf den Markt. Der Sirius I war der erste preiswerte Microcomputer der 16-Bit-Generation, der in großer Auflage produziert wurde und weite Beachtung fand.

Die Bedienung war durch eine frei bewegliche Tastatur und den hochauflösbaren Grafikmonitor mit blendfreiem Bildschirm angenehm einfach. Der Sirius I setzte einen bis dahin nicht gekannten Standard für Microcomputer-Bürosysteme.

Herman Hollerith

1860
Geboren in Buffalo im Staate New York

1879
Universitätsabschluß an der Columbia University und Annahme einer Assistentenstelle bei der amerikanischen Volkszählungsbehörde

1883
Arbeit im Patentamt in Washington

1884
Anmeldung der ersten Patente für die Übertragung von Informationen auf Endlos-Papierstreifen

1887
Sein System wird zum ersten Mal in Baltimore, Maryland, für die Sterblichkeitsstatistik eingesetzt

1889
Das System wird im Büro des Oberstabsarztes der Armee für Krankheitsstatistiken eingesetzt; Hollerith meldet die Lochkarte zum Patent an

1890
Erhält den Zuschlag, mit seinen Maschinen die Auswertung der Volkszählung von 1890 durchzuführen; die Columbia University verleiht ihm den Doktortitel für seine Arbeit über Datenverarbeitung

1901
Die neue Maschinengeneration wird für landwirtschaftliche Statistiken eingesetzt

1911
Gründung einer Dachgesellschaft, der Tabulating Recording Company

1924
Die Firma ändert ihren Namen in International Business Machines (IBM)

1929
Stirbt in Washington DC

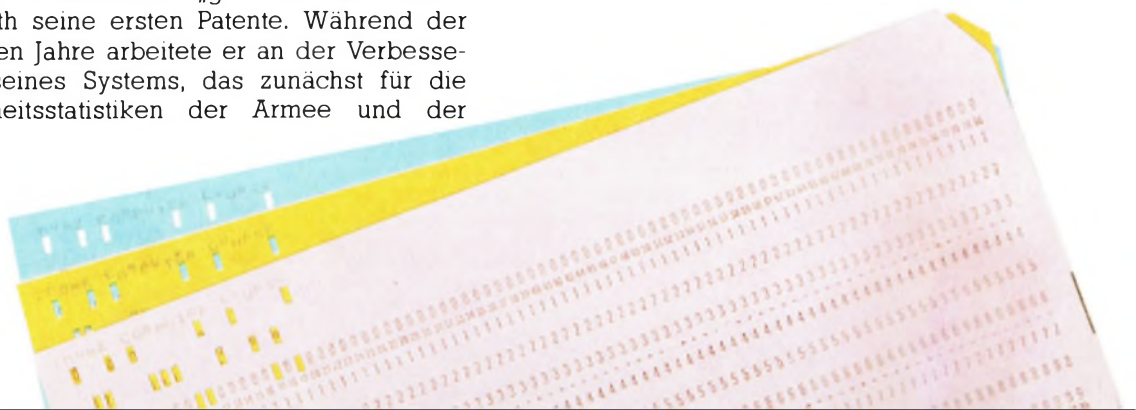
Der geniale Erfinder, der als erster mit Lochkarten arbeitete, gründete eine Firma, die sich zum größten Computerhersteller der Welt entwickelte.

Herman Hollerith wurde 1860 in Amerika geboren. Nach einem Studium an der Columbia University arbeitete er als Assistent in der Volkszählungsbehörde der Vereinigten Staaten und wertete die Daten der Zählung im Jahre 1880 statistisch aus. Da alle Daten mühsam per Hand ausgewertet wurden, lagen bei der nächsten Zählung, zehn Jahre später, noch nicht einmal alle Ergebnisse der letzten Befragung vor. Hollerith glaubte fest daran, dieses Problem lösen zu können, und er ließ sich deshalb von dem National Census Office zum Patentamt in Washington versetzen.

Bei seinem ersten Lösungsversuch übertrug Hollerith die Daten zunächst auf Papierbänder, die in verschiedene „Felder" unterteilt waren. Jedes einzelne symbolisierte eine bestimmte Kategorie wie z. B. männlich/weiblich oder schwarz/weiß. Die kreisrunden Löcher wurden später maschinell „gelesen". 1884 erhielt Hollerith seine ersten Patente. Während der nächsten Jahre arbeitete er an der Verbesserung seines Systems, das zunächst für die Krankheitsstatistiken der Armee und der schnell anwachsenden amerikanischen Städte eingesetzt wurde.

Fünf Jahre später führte Hollerith eine wesentliche Verbesserung ein: Statt eines gelochten Papierstreifens wurde für jede befragte Person eine eigene Karte angelegt. Er konstruierte einen Speziallocher, der 6 mm große quadratische Löcher ausstanzte. Eine einzige Lochkarte konnte auf diese Weise eine Reihe unterschiedlicher Informationen und Daten enthalten.

1889 stellte sich auch der kommerzielle Erfolg ein: Die Volkszählungsbehörde verkündete eine Ausschreibung für Maschinen, die die nächste Zählung statistisch auswerten sollten. Hollerith machte mit Abstand das Rennen. Da alle seine Maschinen patentiert waren, nutzte er seine Monopolstellung und berechnete der amerikanischen Regierung 65 Cents für die Verarbeitung von je tausend Karten. Obwohl es für jede gezählte Person eine eigene Lochkarte gab, benötigte Hollerith für seine Auswertung nur zwei Jahre. Danach stand fest, daß Amerika 56 Millionen Einwohner hatte, und Hollerith sandte der Regierung eine entsprechend hohe Rechnung. Bis zur Volkszählung im Jahre 1900 hatte er seine Maschinen wiederum wesentlich verbessert, weigerte sich aber, seinen Preis zu senken. Als seine Patente schließlich abliefen, suchte die amerikanische Regierung nach anderen Firmen, die diese Arbeit billiger erledigen konnten. Hollerith gründete jedoch eine eigene Firma und schlug auch diesmal die Konkurrenz aus dem Feld. Seine Firma wurde später in „International Business Machines" umbenannt und ist heute mit 20 Milliarden Dollar Jahresumsatz der größte Computerhersteller der Welt.



Noch heute verwendet man Holleriths Methode zur Darstellung von Informationen. Geändert hat sich nur das Format der Karten: Die modernen Lochkarten haben 12 Zeilen mit je 80 Spalten. Tabelliermaschinen arbeiten nach dem Dezimalsystem und können auf jeder Karte 80 Zahlen darstellen. Mit Computern können aber auch binär-codierte Karten verarbeitet werden.

Ändern und Löschen

Die Möglichkeit, Daten auf dem Bildschirm aufzubereiten, wie sie von den meisten Microcomputern geboten wird, hilft viel Zeit und Mühe zu sparen.

Beim Eingeben von Daten in den Computer sind Tippfehler unvermeidlich. Unvermeidlich ist es auch, bereits eingegebene Daten inhaltlich zu ändern, zu ergänzen oder auf den neuesten Stand zu bringen. Wir brauchen darum eine Möglichkeit, auf dem Bildschirm angezeigte Daten „aufzubereiten". Diese Möglichkeit bietet der „Editor" wie die Aufbereitungsfunktion genannt wird.

Den Editor gibt es in irgendeiner Form in den meisten Anwenderprogrammen. In einem Textverarbeitungssystem beispielsweise ermöglicht der Editor eine Textaufbereitung in der Form wie sie beim Überarbeiten von Berichten und Briefentwürfen gebraucht wird.

Bei fast allen Heimcomputern ist im Betriebssystem ein Editor eingebaut, der das Korrigieren und Ändern von Programmlistings ermöglicht. Gerade bei der Programmeingabe passieren häufig Eingabefehler; Syntaxfehler treten mit ziemlicher Regelmäßigkeit auf, und beim Programmablauf ist mit Sicherheit noch der eine oder andere „Mißgriff" zu beseitigen. Dabei darf nicht vergessen werden, daß ein guter Programmierer bereits vor dem Eingeben eine Menge Zeit dafür aufgewendet hat, sein Programm schon auf dem Papier auf eventuelle Logikfehler zu überprüfen. Es ist eine schlechte Arbeitsweise, ein Programm „aus dem Kopf" einzutippen und sich dann erst mit einem vielfach größeren Zeitaufwand auf die Fehlersuche zu begeben.

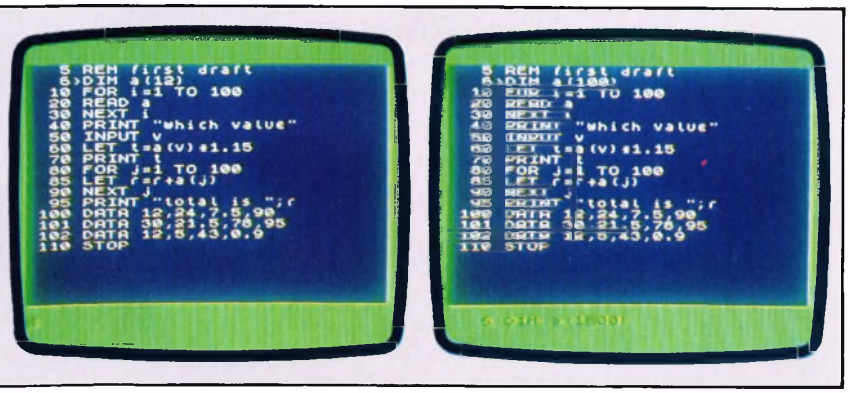
Es gibt zwei Editor-Arten: den Bildschirm-Editor und den Zeilen-Editor. Der Bildschirm-Editor erlaubt ein freizügiges und leichtes Arbeiten; bei zahlreichen Heimcomputern steht jedoch nur der Zeilen-Editor zur Verfügung. Bei diesem hat der Programmierer zwar die Möglichkeit, mit Hilfe der Anweisung LIST das ganze Programm auszudrucken, wenn jedoch in einer Zeile eine Korrektur notwendig wird, muß die ganze Zeile neu eingetippt werden. Bei einigen Systemen kann man mit EDIT und nachfolgender Zeilennummer diese eine Zeile separat auf dem Bildschirm darstellen und dann Änderungen und Löschungen mit den Tasten „backspace" (Rücktaste) und „rubout" durchführen. Andere Befehle wie DELETE machen es möglich, bestimmte Zeilen zu löschen.

Der Bildschirm-Editor dagegen ist flexibler und einfacher zu handhaben. Er gestattet, Text- oder Zeichensymbole ohne Mühe auf dem Bildschirm hin- und herzubewegen. Wird die Taste RETURN betätigt, liest der Editor die Programmzeile, auf der der Cursor steht, in den Interpreter.

Ein Bildschirm-Editor muß in Maschinencode geschrieben sein, um ausreichend schnell zu arbeiten. Ein komfortabler Editor verfügt über zusätzliche Funktionen, wie z. B. das „scrolling", d. h. die Möglichkeit, ein ganzes Programmlisting von oben nach unten oder umgekehrt über den Bildschirm wandern zu lassen. So kann man sowohl ganze Zeilen als auch einzelne Zeichen einfügen oder löschen oder sogar, wie bei einem Textverarbeitungssystem, bestimmte Textpassagen (Namen, Zeichenfolgen usw.) bei jedem Auftreten im Text oder der Liste herausfinden und ändern. Mit jeder neuen Computergeneration werden die Editor-Leistungen vielseitiger und die Handhabung einfacher. Mit der Entwicklung der „Maus" und der Verfügbarkeit von Software, die das manuelle Schneiden und Zusammenstellen von Textteilen „nachahmt", wird der Zeitaufwand immer geringer, den man benötigt, um ein Schriftstück oder eine Liste vom ersten Entwurf bis zur endgültigen Form „aufzubereiten".

Zeile um Zeile

Der Spectrum bietet bessere Editiermöglichkeiten als die meisten anderen Zeilen-Editoren. Um eine bestimmte Zeile in einem Programm zu ändern, muß der Positionszeiger (›), der zwischen der Zeilennummer und der Zeile selbst erscheint, mit Hilfe der Cursorsteuerungstasten zu der entsprechenden Zeile bewegt werden.



Konrad Zuse

Während John von Neumann seine Pionierarbeiten in den USA vollbrachte, arbeitete Konrad Zuse in Deutschland an denselben Problemen.

Zuse entwickelte seine Rechner, um die großen Teams zu entlasten, die bei der Flugkörper-Konstruktion langwierige Kalkulationen mit dem Rechenschieber anstellen mußten. Auch bei der Entwicklung der V 1- und der V 2-Raketen wurden seine Geräte eingesetzt.

Als in den USA gerade der erste relaisgesteuerte Computer (ENIAC) entstand, arbeitete der deutsche Ingenieur Konrad Zuse bereits an einem programmierbaren Rechner – höchstwahrscheinlich dem ersten Computer der Welt.

Zuse, 1910 in Berlin geboren, arbeitete nach dem Studium an der Technischen Universität der Stadt als Luftfahrttechniker bei den Henschel-Flugzeugwerken, wo er neue Tragflächenkonstruktionen entwarf. Die grundlegenden mathematischen Methoden für die Berechnung der Form und Stabilität von Tragflächen waren schon seit den 20er Jahren bekannt. Der Rechenaufwand für derartige Neuentwicklungen war jedoch enorm. Mit der Planung war ein ganzes Team beschäftigt, das sich mit mechanischen Rechenmaschinen und Rechenschiebern abplagen mußte. Zuse suchte Alternativen zu dieser zeitraubenden Arbeit. Zusammen mit Freunden begann er in der elterlichen Wohnung mit dem Bau eines ersten „Computers".

Der Einstieg war ein rein mechanisches Gerät, der „Z1". Damit ließen sich neben den vier Grundrechenarten schon Quadratwurzeln ziehen und dezimale in Binärziffern verwandeln. Die wichtigste Erkenntnis bestand jedoch in der simplen Feststellung, daß ein Hebel jeweils zwei (Schalt-)Positionen einnehmen kann, Ein bzw. Aus. Dadurch eignet er sich sowohl zum „Speichern" von Daten als auch zum Steuern des Rechenvorganges selbst.

Steiniger Weg

Zuse beschritt daraufhin konsequent den Weg, sowohl Daten als auch Rechenanweisungen in rein binärer Form darzustellen, wodurch er 1941 zur Konstruktion der „Z2" fand, seines ersten elektromechanischen Rechners. Nachdem die militärischen Einsatzmöglichkeiten des Gerätes erkannt wurden, stellte die vorher eher uninteressierte Regierung dann auch entsprechende Mittel zur Weiterentwicklung bereit, die dem Bau des „Z3" zugute kamen. Hier waren mechanische Verbindungen erstmals vollständig durch elektrische Verdrahtung ersetzt, wodurch die Maschine viel kleiner und technisch eleganter wurde.

Der Krieg legte Zuse bei seiner Arbeit einen Stein nach dem anderen in den Weg: Zuerst wurde er zweimal an die Ostfront einberufen, dann mußte er wegen alliierter Bombenangriffe mehrfach die Werkstatt wechseln. Zudem gab es kaum geeignetes Material. Ausgeschlachtete Telefone dienten ebenso als Hilfsmittel wie gebrauchte Filmstreifen, die Codestreifen aus Papier ersetzen mußten. Trotzdem speicherte der „Z3" schon 64 Worte à 22 Bit. Daten konnten über eine Tastatur eingegeben und Ergebnisse auf einem Anzeigebrett mit Lämpchen abgelesen werden. Der „Z3" wurde leider zusammen mit seinen Vorgängern bei den Bombenangriffen auf Berlin im Jahre 1945 zerstört. Vorher wurde das Gerät aber auch praktisch eingesetzt: bei der Konstruktion eines unbemannten, fernlenkbaren Bombenflugzeuges.

Der letzte Computer aus Kriegszeiten war der „Z4", der Worte bis zu 32 Bit verarbeiten konnte. Er wurde beim Vorrücken der Alliierten nach Göttingen ausgelagert und später bis zum Jahre 1954 in Basel eingesetzt – lange einer der wichtigsten Computer, die es in Europa zu dieser Zeit gab.

Mit dem Zusammenbruch endete für Zuse vorläufig die Möglichkeit, in Deutschland einen Computer zu bauen. Er wandte sich daher der Rechner-Theorie zu und entwickelte die Programmiersprache „Plankalkül", mit der sich sowohl mathematische als auch allgemeinere Probleme bearbeiten ließen. Eine später von Zuse gegründete Firma blieb bis zur Übernahme durch Siemens Deutschlands bedeutendste Computer-Herstellerin. Heute ist Professor Zuse nur noch wissenschaftlich tätig.

Sternschnuppen

In der Astronomie gibt es zwei Hauptaufgaben für den Computer: Er dient zum Speichern der Daten beobachteter Objekte und richtet das Teleskop gleichzeitig genau aus.

Optische Astronomie wird viel einfacher, wenn eine genaue Vorhersage über den Standort eines Sterns oder Planeten für einen bestimmten Tag vorhanden ist. Der Heimcomputer als Datenbank ermöglicht die Berechnung der Koordinaten in Sekunden.

Wenn Sie den nächtlichen Himmel beobachten und dabei den uns nächsten Stern sehen, erkennen Sie diesen tatsächlich nur so, wie er vor vier Jahren war: So lange ist das Licht von Proxima Centauri zur Erde unterwegs. In eben diesen vier Jahren hat sich der Microcomputer vom seltenen und teuren Hobby zum beinahe selbstverständlichen „Hausgenossen" entwickelt. Gerade in der Astronomie lassen sich die Fähigkeiten des Heimcomputers beim Verarbeiten von Daten, Rechnen und Steuern voll nutzen.

Jeder Hobby-Astronom kennt die drei Probleme, die bei der Beobachtung des Himmels auftreten: die erste Beobachtung eines Himmelskörpers, den Umgang mit den dabei ermittelten Daten sowie deren sinnvolle Analyse. In jedem dieser drei Bereiche hilft der Rechner. Die Aufgaben des Computers beginnen manchmal aber schon bei der Konstruktion des Handwerkszeuges – des Teleskops. Durch mathematische Berechnungen der Linsen und Spiegel werden hier die besten Ergebnisse erzielt. Viele „Himmelsfreunde" hätten schon früher gern ein Teleskop selbst gebaut. Vor der Einführung des Heimcomputers griffen viele lieber auf einen Bausatz zurück, als die schwierigen Berechnungen des Strahlenganges selbst durchzuführen. Diese – oft wochenlange – Arbeit kann man mit dem Computer in wenigen Minuten erledigen.

Positionen festlegen

Bei der Suche nach einem bestimmten Stern am Himmel kann uns der Rechner ebenfalls unterstützen. Himmelskörper sind keine feststehenden Objekte. Durch die Erdrotation beschreiben sie Bahnen am Himmel, so daß ihre Position je nach Tages- und Jahreszeit verschieden ist. Zur Lokalisierung dienen dabei Systeme von Längen- und Breitengraden, wie sie auch in der herkömmlichen Geographie üblich sind. Denkt man sich ein auf die Himmelsfläche projiziertes Koordinatensystem, läßt sich jede Sternenposition durch Angabe von Deklination und Rektaszension unverwechselbar festlegen.

Mit diesen Angaben kann man den Stern dann in einer Karte einzeichnen und aus vielen dieser Karten einen Himmelsatlas zusammenstellen. Für Planeten-, Satelliten- oder auch Kometenbeobachtung sind diese Fixpunkte unverzichtbar. Zusammen mit Werten der Leuchtkraft und spektralen Zusammensetzung der Strahlenemission sowie der Art und dem Alter eines Sterns können diese Daten im Computer gespeichert und verwaltet werden. Dadurch kann der Rechner eine Sternkarte für einen beliebigen Standort und jeden Zeitabschnitt auf den Bildschirm projizieren.

Durch die Erdrotation bewegt sich jeder

Stern innerhalb weniger Minuten aus dem Sichtfeld eines Teleskops, sogar, wenn es einen relativ großen Blickwinkel hat. Wer längere Beobachtungen in einem kleinen Gebiet vornehmen möchte, muß die Position des Teleskops daher ständig justieren. Dazu werden seit Jahren mechanische Antriebe verwendet. Heute stehen auch dem Amateur computergesteuerte Systeme zur Verfügung. Das Teleskop wird auf einer nach Norden gerichteten Achse montiert und von einem kleinen Servomotor genau mit der richtigen Geschwindigkeit bewegt, die nötig ist, um die Erdrotation auszugleichen. Auf der Schwenkachse und dem Teleskop selbst sind Meßfühler montiert, die über ein digitales Signal dem Computer die jeweilige Neigung des Teleskops angeben.

Die Überwachung des Motorantriebs durch den Rechner erlaubt beispielsweise das Verfolgen einer Sternenbahn während der ganzen Nacht, so daß auch Fotografien mit sehr großer Belichtungszeit möglich sind. In den USA ist ein Adapter namens „Celestial Navigator MK II" im Handel, mit dem sich die Verbindung zwischen Teleskop und Rechner über die normalen Schnittstellen bewerkstelligen läßt.

Amerikanische Hobby-Astronomen sind beim Computer-Einsatz aber noch weiter gegangen: Sie nutzen die Möglichkeit der Sprachein- und -ausgabe. Durch den Zuruf „Auf" bzw. „Kuppel drehen" läßt sich die Kuppel des Observatoriums öffnen und bewegen. Synthetische Sprache ist in der Dunkelheit eines Observatoriums ebenfalls eine große Hilfe bei der Datenausgabe durch den Computer: So kann der Rechner etwa Zeitintervalle laut ansagen, um dem Beobachter beim Fotografieren exakt die notwendige Belichtungszeit anzugeben.

Profi-Astronomen verwenden auch Teleskope, die anstelle von Licht Radio- oder Röntgenstrahlen registrieren. Die Größe und das Gewicht dieser Geräte führen zu schwierigen Problemen bei der Handhabung. Das 1964 bei Manchester installierte elliptische 38 x 25 Meter-Teleskop war als erstes mit einem Computer verbunden, der die angewählten Himmelskoordinaten in Steuerungsbefehle für die verschiedenen Antriebsmotoren umwandelte. Dabei wurden Ist- und Sollposition des Teleskops viermal in jeder Sekunde verglichen.

Mit Computern lassen sich auch atmosphärische Einflüsse ausgleichen, die durch veränderte Lichtbrechung bei Feuchtigkeits- und Temperaturdifferenzen entstehen. Auch Verzerrungen des Objektbildes durch Eigenschaften des Teleskops selbst können durch den Computer kompensiert werden.

Im Gegenzug zur Hilfe des Computers in der Astronomie haben Himmelskundler aber auch die Computerentwicklung gefördert: Die Programmiersprache FORTH ist eine Entwicklung des Astronomen C. H. Moore vom Kitt Peak Observatorium aus Arizona, der damit ursprünglich das Radioteleskop und dessen Prozeßdaten steuern wollte.

Musterschüler
Die Leistungen von Schülern und Lehrern der Kettering-Schule in Northamptonshire haben der Schule auf dem Gebiet der Astronomie einen besonderen Ruf eingetragen. Die Kettering-Schüler haben mehrfach als erste das Auftauchen eines neuen Satelliten im Orbit beobachtet.

Dieses Riesen-Teleskop wurde nach der Entdekkung von Radiostrahlung aus entfernten Galaxien in Jodrell Bank errichtet. Mit Radioteleskopen können Objekte ausgemacht werden, die auch dem leistungsfähigsten optischen Teleskop verborgen bleiben.

Grace Hopper

Bei der Entwicklung höherer Programmiersprachen erwarb sich Grace Hopper große Verdienste. Sie wirkte, allerdings unfreiwillig, auch bei der Computer-Terminologie mit.

Die Computer-Wissenschaft galt früher als ausschließlich männliche Domäne. In der Entwicklung und Anwendung ziehen die Frauen jedoch zunehmend gleich auf. Grace Hopper hat als eine der ersten weiblichen Computer-Pioniere Großes auf dem Gebiet der Software geleistet – sie erfand den ersten Compiler und wirkte bei der Einführung der Programmiersprache COBOL mit.

Nach ihrer Doktorandenzeit in Yale kehrte Grace Hopper als Dozentin für Mathematik an ihren ersten Studienort Vassar zurück. Mit 39 Jahren wurde sie an ein Computer-Projekt des Verteidigungsministeriums berufen. 1945 ging Frau Hopper als Assistentin Howard Aikens nach Harvard, um ihn bei der Konstruktion eines Computers zu unterstützen. Aiken hatte schon 1937 für IBM einen mechanischen Rechner entworfen. Dieser arbeitete so gut, daß IBM daraus ein verbessertes Modell, den „Harvard Mark II" entwickeln ließ, der schon mit elektromechanischen Relais arbeitete.

Die frühen Computer wurden noch ausschließlich durch Änderungen in der Verdrahtung für unterschiedliche Aufgaben programmiert. Grace Hopper war im Sommer des Jahres 1945 fast ununterbrochen mit dem Verdrahten des Computers beschäftigt: Für den Krieg wurden dringend Möglichkeiten ballistischer Berechnungen gebraucht, und Aiken kam regelmäßig in die Werkstatt, um neue Ergebnisse zu erfragen. Nach einem Totalausfall des Rechners fand man nach langer Suche einen Nachtfalter, der sich in einem der Relais verklemmt hatte. Auf Aikens Drängen nach neuen Zahlen bekam er von Grace Hopper die lakonische Antwort: „Wir müssen erst die Käfer (bugs) aus der Maschine entfernen!" Bis heute werden Computer-Fehler „bugs", also „Käfer" genannt. Den Nachtfalter, der seinen Platz in der Computer-Sprache mit dem Leben bezahlt hat, kann man heute noch sehen: Er ist in die Dokumentation des „Harvard Mark II" eingeklebt worden – neben der Eintragung vom 9. September 1945, 15.45 Uhr. Das Buch ist im Marinemuseum von Virginia ausgestellt.

Im selben Jahr bauten John Mauchly und Presper Eckert den Computer ENIAC, den sie nach dem Krieg für kommerzielle Zwecke weiterentwickelten. Grace Hopper entwarf große Teile der Software für diesen UNIVAC genannten Rechner. Bei den Programmierarbeiten kam sie darauf, daß es nicht unbedingt notwendig sei, bestimmte Unterprogramme immer wieder neu einzugeben. Grace Hopper schuf zusammen mit der Programmiersprache den ersten „Compiler", der das Programm in die Maschinensprache übersetzen konnte.

COBOL erfunden

Grace Hoppers Neuerung erhielt den Namen „A-O". Als der Compiler erstmals vorgestellt wurde, war die Fachwelt überrascht, denn bisher hielt man nur Rechenoperationen oder den Umgang mit Symbolen für möglich. Nun aber sprang der Rechner auf einen einfachen Befehl hin direkt zu einem Unterprogramm im Speicher, und die Befehle ähnelten schon einem normalen englischen Satz.

Im Mai 1959 wurde Captain Grace Hopper vom Pentagon dazu eingeladen, sich an einer Arbeitsgruppe zu beteiligen, die eine einheitliche Computersprache für kommerzielle Anwendungen entwickeln sollte. In weniger als einem Jahr entstand so die „COmmon Business Oriented Language" (COBOL). Grace Hopper übernahm jeweils das Beste aus den schon bekannten Computersprachen und optimierte es so weit, daß COBOL sich in der Geschäftswelt schnell durchsetzen konnte. Ein Beweis für die hohe Qualität der Sprache ist, daß sie bis heute eine der meistgebrauchten Programmiersprachen insbesondere für die kommerzielle Anwendung geblieben ist.

COBOL

COBOL war eine der ersten Programmiersprachen, die besonders für mathematische Laien gedacht waren. Schwierige Codes werden bei COBOL durch das Abrufen allgemeinerer Abläufe mit sprachähnlichen Befehlen ersetzt. Ein COBOL-Programm besteht aus vier Einheiten: Der Programmname, der Name des Autors und Zusatzinformationen bilden die Abteilung „Identifikation". Obwohl COBOL-Programme auf unterschiedlichen Rechnern laufen sollen, werden im Bereich „Environment" (Umgebung) spezielle Daten über den Rechner abgelegt, auf dem das Programm geschrieben wurde. Da oft dieselben Daten in verschiedenen Programmteilen genutzt werden, wird in COBOL außerdem eine spezielle DATA-Abteilung eingerichtet. Im PROCEDURE-Bereich stehen endlich die Operationen, die auf die Daten angewendet werden sollen – also der Programm-Ablauf.

Mark I-Triumph

An der Hochschule von Manchester wurde der erste programmierbare Rechner der Welt entwickelt.

Kurz nach Ende des Zweiten Weltkrieges stellte die Universität Manchester zwei neue Professoren ein: Max Newmann, der vorher am ersten elektromechanischen Computer „Colossus" in Bletchley Park mit Decodieraufgaben betraut war, erhielt den Titel „Professor der Mathematik". Der Radartechniker F. C. Williams wurde neuer Leiter des Bereichs Elektrotechnik. Sein Assistent Tom Kilburn, der ihn nach Manchester begleitete, hatte im Krieg bereits an elektronischen Speichereinheiten für Radaranlagen gearbeitet. Kilburn wurde später erster Professor der neuen Abteilung „Computer-Technik".

Während seiner Studienreise in die Vereinigten Staaten wurde Williams 1946 ein Prototyp des Röhren-Computers „ENIAC" vorgeführt. Nach seiner Rückkehr brachte er die englische „Royal Society" dazu, 35 000 Pfund in das neue Labor für Rechenmaschinen zu investieren. Im Rennen um neue Computer, die ein Programm speichern konnten, stand seine Hochschule nicht allein: In Pennsylvania wurde bereits am EDVAC, in Cambridge am EDSAC gearbeitet. Auch das Nationale Physikalische Labor beteiligte sich mit seinem ACE-Programm. Das Team der Universität von Manchester verwendete jedoch einen von Williams konstruierten Speicher mit Kathodenstrahlröhre, während die Konkurrenten mit Quecksilberröhren arbeiteten. Bereits im Herbst 1947 erreichte man, daß 2048 Bits über mehrere Stunden gespeichert werden konnten.

Erstes Programm 1948

Auf dem „Manchester Mark I", dem ersten programmierbaren Computer der Welt, lief dann 1948 das erste Programm. Für einen Rechenschritt brauchte der „Mark I" nur 1,2 Millisekunden. Durch die Verwendung der Kathodenstrahlröhre (CRT) konnte der Inhalt des Hauptspeichers und der verschiedenen Register dargestellt werden. Zudem war der Zugriff auf einzelne Speicherzellen unabhängig vom Zustand der anderen Zellen.

Nachdem die Brauchbarkeit erwiesen war, wurde der Mark I für optische Berechnungen und zur Primzahl-Erzeugung weiterentwickelt. Die Vorführung dieses Gerätes beeindruckte den wissenschaftlichen Inspektor der Regierung Sir Ben Lockspeiser so sehr, daß er für die kommerzielle Produktion durch eine Firma in Manchester sorgte. Fünf Monate vor der Fertigstellung des UNIVAC-Rechners konnte man dann im Februar 1951 mit dem „Ferranti Mark I" den ersten kommerziellen Computer der Welt kaufen.

Befehlsänderungen möglich

Neu am Mark I war die Möglichkeit, Befehle während des Rechenvorganges zu ändern und in den Programmspeicher zu übertragen – die Verarbeitungsgeschwindigkeit nahm zu. Für ihre ersten Rechner übernahm später IBM einige der Manchester-Patente. Als Williams das IBM-Gebäude in New York besuchte, stieß er überall auf das Firmenmotto „THINK" (DENKE). Später wurde ihm die Frage gestellt, wie sein Team die Aufgaben lösen könnte, bei denen selbst IBM passen mußte. Ein schnelles „Wir haben uns eben nicht die Zeit genommen, viel zu denken" war seine Antwort.

1948 kam Alan Turing, der schon 1959 das erste Handbuch zum Programmieren herausgeben sollte, nach Manchester. Zwei Jahre später plante die Gruppe dann einen kleinen, leistungsfähigen Rechner, ein Vorhaben, das durch die Einführung des Transistors vorangetrieben wurde. Im November 1953 arbeitete in Manchester der erste Transistor-Computer der westlichen Welt.

In den späten 50er Jahren versuchte die englische Industrie noch einmal, den Amerikanern die Spitzenposition in der Computertechnologie streitig zu machen. Tom Kilburns Team entwickelte mit Unterstützung der Regierung den „Atlas". Der Rechner mit 48-Bit-Worten, 16 KByte-Speicher und einem 8 KByte-Trommelspeicher war seiner Zeit weit voraus und blieb lange ein Spitzenreiter auf der „Computer-Hitliste". Sogar die British Petroleum (BP) und das Atomenergie-Forschungszentrum in Harwell arbeiteten lange damit.

Das erste Programm lief im Juni 1948. Damit war der Manchester Mark I der erste programmierbare Computer. Die damals noch kleine Firma Ferranti erhielt den Auftrag, aus dem Mark I eine kommerzielle Version zu entwickeln. Diese kam im Frühjahr 1951 auf den Markt.

Alan Turing

Der britische Mathematiker entwickelte einen allgemein anerkannten Test zur Feststellung künstlicher Intelligenz. Während des Zweiten Weltkrieges arbeitete er als Codeknacker für den Geheimdienst.

Schon in sehr jungen Jahren zeigte Alan Turing eine bemerkenswerte Begabung für Naturwissenschaften. Er hatte gerade erst Lesen und Schreiben gelernt, als er bereits magische Zahlen in Primfaktoren zerlegte und einen Entwurf für ein „schwimmfähiges Fahrrad" erstellte. Jahre später, während sein Vater im fernen Madras bei der indischen Verwaltung tätig war, erhielt Alan ein Stipendium für das King's College in Cambridge. Sein ganzes Interesse konzentrierte sich nunmehr auf die Probleme mathematischer Logik – zunächst noch als Student, später als wissenschaftlicher Mitarbeiter der dortigen Hochschule.

Im Jahre 1931 verblüffte der tschechische Mathematiker Kurt Gödel die Fachwelt mit der Entdeckung wahrer, aber nicht beweisbarer mathematischer Lehrsätze. Alan Turing begann jedoch mit der Untersuchung beweisbarer Lehrsätze. Er entwarf das Modell einer Maschine, das auf mechanische Weise Operationen durchführen konnte – Arbeiten, die sonst von Mathematikern verrichtet werden. So entstanden Geräte zum Addieren, zum Integrieren, für die Division und für andere Rechenarten. Die Denkmodelle wurden später unter der Bezeichnung „Turing-Maschinen" bekannt.

Turing untersuchte die Funktionsweise dieser imaginären Geräte und kam zu einem bemerkenswerten Schluß. Statt für jede mathematische Operation einen eigenen Rechner bereitzustellen, mußte es möglich sein, einen „Universalrechner" zu entwerfen, der durch entsprechende Programmierung jeden „Spezialrechner" ersetzen kann.

Als der Zweite Weltkrieg ausbrach, wurde Turing von der Universität zur „Government School of Codes and Ciphers" in Bletchley Park, Buckinghamshire, eingezogen. Ohne den Krieg wären seine Maschinen vielleicht nur trockene Theorie geblieben, aber Bletchley Park war mit der dringenden und streng geheimen Aufgabe beschäftigt, die deutschen Militär-Codes zu knacken.

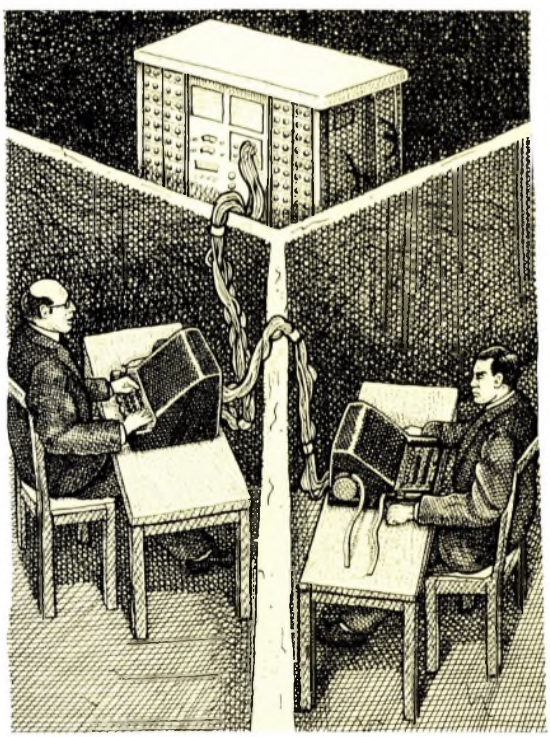
Weil diese jeden Tag geändert wurden, benötigte man Maschinen, um den jeweils neuesten Schlüssel zu finden. Bletchley Park wuchs zu einem riesigen Zentrum für Informationsverarbeitung. Alan Turing wurde während des Krieges auch nach Amerika geschickt, um für die transatlantischen Nachrichtenverbindungen der Alliierten Streitkräfte sichere Codes zu entwickeln.

Da seine Arbeit zu dieser Zeit geheim war, sind nur wenige Berichte über seine Reisen vorhanden. Es wird jedoch allgemein angenommen, daß er während seines Aufenthalts in Princeton, New Jersey, mit John von Neumann zusammentraf. Gegen Ende des Krieges wurde Turing gebeten, einen britischen Computer mit Namen ACE für das National Physical Laboratory zu entwickeln.

Die „Automatical Computing Engine" (Automatisch rechnende Maschine) wurde zu Ehren von Charles Babbage „Analytical Engine" benannt. Wie bei diesem Vorläufer der Computertechnik dauerte auch der Bau von ACE sehr lange, aber der Rechner war dem amerikanischen ENIAC in vielerlei Hinsicht weit voraus. Enttäuscht vom schleppenden Fortgang der Arbeiten ging Turing resigniert nach Manchester, wo er sich am Computerprojekt der Universität beteiligte. Zur gleichen Zeit wurde er Berater der Ferranti-Gesellschaft und beschäftigte sich in der Folgezeit mit den ersten Computern, die in Großbritannien gebaut wurden.

Ein Freund sagte einmal, daß Turing, was die Fehler bei anderen angehe, eher großzügig verfahren sei, sein eigenes wissenschaftliches Genie dagegen war unbestritten. 1952 wurde er wegen homosexueller Delikte verurteilt und beging zwei Jahre später Selbstmord.

Alan Turing (1912–1954) fand beim Langstreckenlauf Entspannung. Er war interessiert an den Auswirkungen körperlicher Anstrengung auf die Kreativität und die geistige Beweglichkeit.



Turing entwickelte das „Nachahmungsspiel", das später als Turing-Test bekannt wurde. Eine Person wird in einen Raum gebracht, in dem sich ein Fernschreiber befindet. Dieser ist mit einem weiteren Fernschreiber im Nebenraum verbunden, der ebenfalls von einer Person oder von einem Computer bedient wird. Die erste Person stellt eine beliebige Frage. Wenn nicht eindeutig zu bestimmen ist, wann mit einem Menschen und wann mit der Maschine kommuniziert wurde, kann die Maschine als intelligent bezeichnet werden.

Norbert Wiener

Die mathematischen Studien dieses genialen Mannes bildeten den Grundstein für das neue Forschungsgebiet „Kybernetik".

Norbert Wiener wurde 1894 in Missouri, USA, geboren. Das Diplom der Mathematik erhielt er schon als 14jähriger, mit 18 war er bereits Professor für Logik. Wiener war auch in Deutschland tätig: als Gastprofessor unter dem berühmten Mathematiker David Hilbert an der Universität Göttingen.

Zur Computer-Entwicklung kam Wiener erst recht spät in seinem Leben. Zuvor hatte er lange Jahre am Massachusetts Institute of Technology (MIT) die Bewegung von Festkörpern in Flüssigkeiten (Brownsche Bewegung) studiert. Der Bewegungsablauf von Teilchen war mit den klassischen Methoden der Physik nicht vorauszusagen, so daß eine neue Berechnungsweise entwickelt wurde. Damit konnte immerhin die Wahrscheinlichkeit einer bestimmten Konstellation von Teilchen zu einem vorgegebenen Zeitpunkt berechnet werden.

Beim Ausbruch des Zweiten Weltkriegs stellte sich Wiener in den Dienst der US-Regierung und widmete sich den mathematischen Problemen, die das Erfassen bewegter Objekte ermöglichten. Die Entwicklung automatischer Zielvorrichtungen, seine physikalischen Studien und breitgestreute Interessen von der Philosophie bis zur Neurologie verarbeitete er 1948 in dem Buch „Kybernetik".

Kybernetik ist die Wissenschaft von sich selbst regulierenden, stabilen Systemen, die man in der Mechanik ebenso findet wie in der Biologie oder Elektrotechnik. Wiener stellte die Information in eine Reihe zu den Faktoren Energie und Materie: Kupferdraht beispielsweise läßt sich sowohl auf die transportierte Leistung wie auch auf die übermittelte Information hin untersuchen.

Der Begriff Kybernetik stammt aus dem Lateinischen und bedeutet „Steuermann". Wiener hatte die Funktionen von James Watts Dampfmaschinen studiert und dabei erkannt, daß Computer eine besondere Fähigkeit der Selbstregulation erfordern, um richtig zu funktionieren.

Negative Rückkopplung

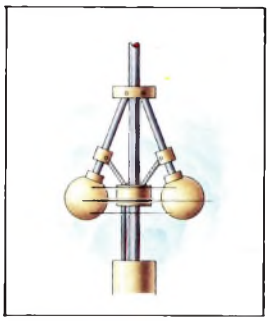
Beispiel für eine solche Regelung ist etwa ein Thermostat: Er steuert die Heizung in einem Hause je nach dem Über- oder Unterschreiten einer Temperatur. Der Mensch stellt nur noch den gewünschten Wert ein. Diese Fähigkeit zur Selbstüberwachung bezeichnete Wiener als „negative Rückkopplung" – Rückkopplung, weil der vom System erzeugte Effekt (die Wärme) das zukünftige Verhalten des Systems beeinflußt, negativ, weil das Schalten des Thermostates die Heizung auf die eingestellte Temperatur drosselt.

Es gibt auch Systeme mit „positiver Rückkopplung". Sie würden – in unserem Beispiel – auch die Temperatur selbst wählen. Ein Automat, der sich neben der Auswahl seiner „Ziele" auch noch selbst reproduzieren könnte, wäre schon beinahe perfekt.

Wieners kybernetische Theorie hat die Forschung im Bereich Information und Regelung stark erweitert. Alles kann als Information betrachtet werden, denn wir erkennen die Umwelt mit Ohr, Auge und Tastsinn. Jeder dieser „Sensoren" beschränkt unsere Wahrnehmung auf eine vergleichsweise winzige Bandbreite des eigentlichen Geschehens, das uns in seiner Gesamtheit überfordern würde. Die Analyse von Zusammenhängen dieser eingeschränkten Sicht gehört ebenfalls zum Bereich der Kybernetik. Aber auch das Entschlüsseln von Daten mittels statistischer Methoden rechnet die Wissenschaft dazu.

Norbert Wiener starb 1964, also lange vor der Microcomputer-Revolution. Dennoch hat er viele der heute auftretenden Probleme vorausgesehen und auch darüber geschrieben.

Wiener war vom Fliehkraftregler der ersten Dampfmaschinen als einfachstem Beispiel für negative Rückkopplung fasziniert. Dabei sind zwei Gewichte an Hebelarmen befestigt, die auf einer mit dem Schwungrad der Maschine gekoppelten Achse sitzen. Bei zunehmender Drehzahl wandern die Gewichte nach außen und schließen über eine mechanische Verbindung die Drosselklappe. Die Drehzahl stabilisiert sich beim eingestellten Wert. In modernen Rechnern ist diese Rückkopplung sehr viel komplexer, das Prinzip bleibt jedoch gleich.



Vannevar Bush

Sein elektromechanischer Computer konnte auch schwierige Differentialgleichungen lösen.

Der Differential-Analysator

Diese Maschine wurde für die Lösung einer wichtigen Gruppe mathematischer Probleme gebaut, die in Technik und Wissenschaft häufig vorkommen – die Differentialgleichungen zweiter Ordnung. Die zuerst von Lord Kelvin vorgeschlagene Methode dazu bestand in der Eingabe der Ergebnisse eines „Integrierers" (Gerät, das die Fläche unter einer Kurve bestimmt) in einen zweiten. Vor der Erfindung von Verstärkern war dies jedoch wegen der zu schwachen Ausgangssignale eines Integrierers nicht möglich. Bushs 1931 entwickelte Maschine war vollständig mechanisch aus komplizierten Getrieben, Achsen und Elektromotoren aufgebaut. Ein- und Ausgabe wurden durch Rotation bewerkstelligt, das Problem der Rückkopplung durch einen Drehmomentverstärker beseitigt. In den 40er Jahren wurde ein verbesserter Differential-Analysator mit elektrischen Komponenten gebaut, der jedoch mehr als hundert Tonnen wog. Die Ergebnisse erschienen in digitaler Form und waren auf 5 Stellen genau. Die Maschinensteuerung ging mit Lochstreifen vonstatten. Während des Zweiten Weltkriegs wurde das Gerät für die Verschlüsselung und außerdem für ballistische Berechnungen eingesetzt.

Von vielen Fachleuten wird Vannevar Bush als „Vater des Computers" bezeichnet. Er hatte schon im Jahre 1931 mit dem Differential-Analysator ein mechanisches Gerät entwickelt, das die Forschung auf den richtigen Weg zu den heutigen digitalen Rechnern brachte.

Bush wurde am 11. März 1890 in Boston, Massachusetts, geboren. Wie schon sein Vater, studierte auch der junge Bush die Ingenieurwissenschaften und erhielt 1913 sein Diplom. Nach einer kurzen Beschäftigung bei General Electric wurde er Lehrbeauftragter an seiner ehemaligen Hochschule, bildete sich aber später in Harvard und am Massachusetts Institute of Technology (MIT) weiter. Während des Ersten Weltkrieges war Bush an der Entwicklung von U-Boot-Aufspürgeräten für die US-Marine beteiligt.

Schon als Student machte Bush seine erste Erfindung: ein Gerät zur Landvermessung. Der Mechanismus war in einem Fahrradrahmen aufgehängt und zeichnete während der Fahrt ein Höhenprofil des Geländes. Die neuartige „Vermessungsmaschine" enthielt einen sogenannten „Integrierer", da für die Höhenbestimmung an jedem Punkt alle vorher registrierten Werte berücksichtigt werden mußten.

Bush wurde Dozent für elektrische Energieübertragung am MIT und beschäftigte sich dort intensiv mit einem schwerwiegenden Problem der Energieversorgung: Stromausfälle durch unvorhergesehene, plötzliche Verbrauchsspitzen sollten verhindert werden. Die mathematischen Grundlagen für die Behandlung dieses Problems hatte der schottische Wissenschaftler C. Maxwell mit den nach ihm benannten Gleichungen schon im 19. Jahrhundert gelegt. Der Rechenaufwand für die Vielzahl der zu lösenden Gleichungsaufgaben war jedoch für reine „Handarbeit" zu groß, so daß Bush sich an die Entwicklung einer Maschine für diesen Zweck machte. Dabei konnte er sich auf die Arbeiten von Lord Kelvin stützen, der schon eine Vielzweck-Maschine für die mathematische Vorausberechnung von Ebbe und Flut entworfen hatte.

Der „Produkt-Telegraf"

Erster Erfolg der Forschung war der „Produkt-Telegraf". Mittels eines Potentiometers wurde eine Wellenform in elektrische Spannung umgewandelt, die einem eigens konstruierten Wattmeter (Stromzähler) zugeführt werden konnte. Dieses wiederum „berechnete" ein „Produkt" aus Spannung und Strom.

Der Erfolg des Produkt-Telegrafen bei der Lösung von Gleichungen spornte den Forscher an, sich an ein Gerät für die Bewältigung von Differentialgleichungen zweiter Ordnung zu wagen. 1931 entstand der erste Differential-Analysator, der in Europa und England mehrfach nachgebaut wurde. Sogar die amerikanische „Moore School of Electrical Engineering" – an der später der ENIAC-Computer gebaut wurde – bestellte ein Exemplar. Die Rechengenauigkeit des Produkt-Telegrafen lag nur bei 2%, mit dem neuen Differential-Analysator konnten aber Ergebnisse auf 0,05% exakt ermittelt werden. Die um nicht einmal zwei ganze Stellen verbesserte Genauigkeit verhundertfachte allerdings bei mechanischen Geräten deren Preis – eine ähnliche Verbesserung eines heutigen Digitalrechners ist höchstens doppelt so teuer.

Bush wurde Dekan der Ingenieurschule und 1939 auch Vizepräsident des Carnegie-Institutes. Als Chef der mit Finanzmitteln reichlich ausgestatteten Militärforschung war Bush auch maßgeblich am Manhattan-Projekt beteiligt – der Entwicklung der amerikanischen Atombombe. 1955 zog es ihn aus dem Beruf ins Privatleben. Vannevar Bush starb 1974.

Spiel um Millionen

Wie so viele andere Heimcomputer-Unternehmen war auch Atari das geistige Kind eines Einzelnen. Nolan Bushnell konnte jedoch wohl kaum voraussehen, welche Konsequenzen es haben würde, als er einen Microprozessor-gesteuerten Regler mit einem Fernseher verband und das Spiel „Pong" erfand.

Nolan Bushnells Erfolgsrezept gründete sich darauf, daß er dem Zuschauer eine Kontrollmöglichkeit über die Ereignisse auf dem Bildschirm gab. Er setzte somit den Bedarf an aktiver Fernsehfreizeit in die Wirklichkeit um und förderte die Phantasie von Millionen Jugendlicher. (Das Taschengeld „förderte" er natürlich auch – aus den Taschen.)

Bushnell und seine beiden Partner Ted Dabney und Larry Bryan investierten umgerechnet jeweils etwa 400 Mark, um „Pong" zu starten. 1972 wurde das Spiel im kalifornischen Sunnyvale vorgestellt und erwies sich bald als profitables Geschäft. Ataris Dominanz im Heimvideospiel-Markt resultierte aus einer wenig später getroffenen Entscheidung, Nutzungsrechte für Bushnells Erfindung zu erwerben.

Bevor sich der Publikumsgeschmack von den Videoautomaten auf die Heimcomputerspiele verlagerte, war Atari Marktführer in diesem Bereich. Das war in den siebziger Jahren. Das Vermarkten von Spielen ist identisch mit dem von Schallplatten: Die potentiellen Stars müssen im Mittelpunkt stehen und beworben werden. So lag es nahe, daß Atari vom Medien-Multi Warner Communications International, bekannt für seine Schallplatten- und Film-Aktivitäten, übernommen wurde. Mit dem Videoautomaten-Geschäft machte Atari Ende der siebziger Jahre noch satte Gewinne. Doch 1983 verzeichnete das Unternehmen bereits Einbußen von über 25% Umsatz und bescherte der Muttergesellschaft beachtliche Verluste.

Das wohl bekannteste aller Computerspiele, „Space Invaders" (von Taito), wurde von Atari erworben und erfolgreich vermarktet. Es entwickelte sich zu einem Phänomen und eröffnete ein ganzes Universum intergalaktischer Ballerspiele. In der Zeit des Videoautomatenbooms Ende der siebziger Jahre drehte sich alles um Atari. Ein Renner folgte dem anderen: „Asteroids", „Battlezone", „Centipede", „Lunar Lander", „Missile Command" und „Tempest".

Doch so dramatisch wie der Videoautomatenboom begonnen hatte, endete er auch wieder. Die Kunden wandten sich den Heimcomputern zu, weil sie zwei wesentliche Vorteile boten: Man konnte dieselben Spiele spielen, die es in den Spielhallen gab, doch das Spielen war preiswerter. Und außerdem verfügte man über einen vielseitigen Computer.

Ataris erste Antwort auf diese Marktveränderung bestand darin, die Arcadenhits in Heimcomputer-Variationen herauszubringen. Dabei verwendete man festprogrammierte Steckmodule. Diese Art der Computer-Software erwies sich als sinnvoll, da der Spieler das Programm nicht erst von Diskette oder Cassette laden mußte. Die Festprogrammierung ist jedoch recht teuer. Da die Cartridges nicht frei programmierbar waren, saß das Unternehmen häufig auf Bergen elektronischen Abfalls, bestehend aus unpopulären Spielen.

Sinkendes Glück

Ataris Marketing-Entscheidungen erwiesen sich bald als mangelhaft. Die Verkaufszahlenhochrechnungen basierten auf so erfolgreichen Spielen wie „Pac-Man". Der Preis für die

Ataris Erfolg war auf die Inspiration eines Mannes zurückzuführen – Nolan Bushnell. Als Bushnell 1971 das erste Computerspiel (Pong) erfand, mag er wohl kaum die Tragweite seiner Schöpfung erkannt haben.

Fehleinschätzung des Marktes war hoch. Anläßlich einer Inventur wurden die nicht verkauften Steckmodule (Verkaufspreis zwischen acht und 25 Dollar) auf 14 Lkw verfrachtet und in ein großes Loch in der Wüste von Nevada gekippt.

Atari verpaßte auch eine andere Gelegenheit, die Besonderheiten von Computerprogrammen zu nutzen: Computer-Codierungen müssen nicht unbedingt auf Datenträgern in den Verkauf gebracht werden. Man kann Pro-

Das Atari Videospiel-System (VCS) einschließlich zweier Joysticks, Netzteil und Spielcassette eignet sich nur zum Spielen. Es kann nicht als Computer verwendet werden. Die Programme sind als Steckmodule erhältlich.

gramme mittels Telefon oder über Kabel, über Rundfunk und Fernsehen übermitteln. Neue Techniken und Produkte standen dafür zur Verfügung. So stellte beispielsweise 1983 die Romox Corporation in den USA eine Maschine mit der Bezeichnung „Romox Programming Terminal" vor. Diese 15-Megabyte-Hard-Disk-Maschine konnte durchs Telefonnetz Programme überspielen.

Alternativ zu dieser Methode wurde „Gameline" entwickelt, das Bill van Meister in den USA vorstellte. Gameline bot ein Modem für das Atari VCS an, mit dem Heimcomputer ans Telefonnetz angeschlossen werden konnten. Die Spiele wurden mittels telefonischer Übertragung eingelesen, wobei für 45 Minuten Spielzeit ein Dollar zu zahlen war.

In den siebziger Jahren verdiente Atari Unsummen mit Videoautomaten dieser Art. Nach Auftauchen der Heimcomputer mußte die Marketing-Strategie des Unternehmens jedoch völlig neu überdacht werden.

Zwei der größten amerikanischen Datenbanken, „Compuserve" und „The Source", bieten Spielprogramme als Bestandteil ihres Service an. Auch hier erfolgt die Programmübermittlung mittels Modem und Telefonnetz an bzw. in den Computer. „Coleco", vor Jahresfrist noch Hersteller von Videospielsystemen, hatte ein entsprechendes Abkommen mit AT&T (American Telephone and Telegraph) geschlossen, um einen solchen interaktiven Service bieten zu können. Vor der Übernahme Ataris durch Jack Tramiel hatte das Unternehmen einen Nutzungsvertrag mit Activision vereinbart und außerdem ein vergleichbares System vorbereitet, hausintern als Ataritel bezeichnet, zu dem Computerbesitzer über die Warner Amex Cable Communications Zugriff haben sollten.

Ataris Probleme lagen vornehmlich darin, daß der Videospielmarkt zurückging. Das hat man durch Erweiterung des Heimcomputermarktes auszugleichen versucht. Die Geräte zeichnen sich durch überdurchschnittliche Grafikeigenschaften und leicht anwendbare Software aus. Drei Chips sind Kernstücke der Atari-Rechner: Pokey, Antic und GTIA, die die Ein-/Ausgabe, Grafik und Farbe steuern.

Alle Atari-Rechner der bisherigen Generation basieren auf dem 6502-Microprozessor. Für diese Systeme steht eine große Software-Bibliothek zur Verfügung, darunter VisiCalc, Atarischreiber und eine Finanzverwaltung für den Hausgebrauch. In den USA gibt es außerdem eine Z80-Softcard, womit Digital Researchs CP/M auch auf einem Atari-Computer laufen kann.

Im vergangenen Jahr setzte das Unternehmen seinen Schwerpunkt auf Software. Für England wurde eigens ein Software-Unternehmen verpflichtet, um die hauseigenen Programme auf andere Systeme umzusetzen und zu vermarkten. Dabei ging es vornehmlich um Commodore-Rechner. Wichtiger aber: Man suchte nach jungen englischen Programmierern, um deren Talente zu fördern und für eigene Entwicklungen zu nutzen. Aus heutiger Sicht betrachtet, könnte Atari den Wiederaufstieg schaffen.

Digital Research

Seit seiner Einführung ist das Control Program for Microprozessors (CP/M) zum Industriestandard unter den Betriebssystemen geworden. Der phänomenale Erfolg von CP/M hat das Leben seines Entwicklers Gary Kildall völlig verändert. Er gab seinen Beruf als Dozent auf und gründete ein eigenes Unternehmen, Digital Research.

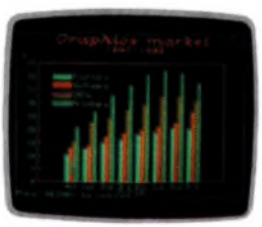
Gary Kildall schrieb 1972 seine erste CP/M-Version, um einen Compiler (ein Übersetzungsprogramm) für PL/M zu schaffen. 1975 schrieb er einen Editor (ED), einen Assembler (ASM) und einen Debugger (Korrektur- bzw. Fehlersuchprogramm) (DDT). Er bot das neue Betriebssystem Intel zur Nutzung an, doch das Unternehmen lehnte ab. Das war das Beste, was Kildall passieren konnte. Gemeinsam mit Dorothy McEwan begann er, eine Reihe von Computermagazinen für Hobbyisten zu publizieren und verkaufte CP/M privat. Kildalls CP/M erzielte bald bessere Erfolge als die Magazine.

Ob es nun an der Programmart lag oder reine Glücksache war: Kildall hatte mit diesem Betriebssystem genau das geschaffen, was eine der „Kinderkrankheiten" der Microcomputer „heilte", nämlich die mangelnde Kompatibilität. Die Ende der siebziger Jahre wichtigsten Computer für den privaten Bereich (PET, Apple und Tandy) waren mit inkompatiblen Diskettenbetriebssystemen ausgestattet. Software-Anbieter mußten sich für eines der Systeme entscheiden. Der Code mußte jeweils völlig neu erstellt werden, um Software auf einer anderen Maschine lauffähig zu machen. CP/M änderte all das: Seine Popularität war Grund für die meisten Hersteller, es zu adaptieren. So wurde ein echter Standard geschaffen. Viele Computerhersteller, die für den Bau ihrer Rechner den Microprozessor 8080 von Intel oder den Zilog Z80 verwendeten, entschieden sich wegen der einfachen Handhabung und des unkomplizierten Zugriffs zu Bildschirm, Drucker, Diskettenstation und Keyboard für CP/M. Mit wachsender Popularität war immer mehr CP/M-Software verfügbar, womit gleichzeitig immer mehr Firmen dieses Betriebssystem in ihre Rechner integrierten.

Schon 1976 war Kildall vom Erfolg seines Produktes und der Nachfrage so überwältigt, daß er seine Professur als Computerwissenschaftler an einer Marinehochschule in Monterey aufgab und in Pacific Grove, Kalifornien, die Firma Digital Research gründete.

CP/M fand weiteren Zuspruch. Indessen widmete sich Digital Research den Mehrplatz-Systemen und entwickelte MP/M. Es war in jeder Hinsicht mit CP/M kompatibel, hatte aber nicht denselben Erfolg. Die Aufteilung der Anwenderbereiche und andere Konfigurationen, die für einen Systemprogrammierer sinnvoll sein mögen, waren keineswegs einfach in der Handhabung. In manchen Bereichen wich die Arbeitsmethode von der bei CP/M auch ab. Die rein technischen Kosten für die Herstellung von Microprozessoren sind bei gleichzei-

Digital Research entwickelt Sprachen und ist mit DR LOGO Marktführer. Wie bei allen guten LOGO-Programmen ist auch hier Grafik der wichtigste Faktor.



GSX ist ein bahnbrechendes Software-Paket, das Grafikprogramme zwischen unterschiedlichen Rechnern übertragen kann, so die hier gezeigten Geschäftsgrafiken.

John Rowley, Präsident von Digital Research Incorporated, rechts Gary Kildall

Der Firmensitz von Digital Research Incorporated ist in Massachusetts, USA.

tig steigender Produktion erheblich gesunken. Daher sank auch der Bedarf für die gleichzeitige Nutzung eines Prozessors durch mehrere Anwender merklich. Somit wurde das Ganze unwirtschaftlich.

1981 erhielt Digital Research eine Finanzspritze durch Risiko-Kapitalgesellschaften und weitete das Unternehmen zu einem echten Multi aus. Das Unternehmen ist in Europa besonders weit verbreitet. Hier gibt es Niederlassungen in England, Deutschland und Frankreich. Ungefähr zur selben Zeit war Digital Research als eines der ersten Häuser darum bemüht, einen Vertrag für die Entwicklung eines Betriebssystems für den neu entwickelten Personal Computer von IBM zu bekommen. Zwar erhielt Microsoft den IBM-Auftrag, doch Digital Research war keineswegs geschlagen. Das CP/M für die Intel-8088/8086-Prozessoren wurde überarbeitet und dem MS-DOS angepaßt.

Entwicklungen nur mit C

Eine der strategischen Entscheidungen bei Digital Research, aber auch vieler anderer System- und Sprachentwicklungshäuser, war, die gesamte Entwicklungsarbeit in der C-Sprache durchzuführen. Das ist für die Übersetzung besonders wichtig. In C geschriebene Codierungen müssen, um auf anderen Microprozessoren zu laufen, lediglich neu kompiliert werden. Allerdings brachte dies den Vorwurf ein, die Codierung sei sehr schwerfällig. Die Methode hat mittlerweile ungeheuer an Popularität gewonnen, und da das weit verbreitete UNIX-Betriebs-System auch in C geschrieben wurde, scheint der Trend zu dieser Sprache unaufhaltsam zu sein.

Heute bietet Digital Research eine Vielzahl von Sprachen für eine Reihe von Microcomputern an. Der untere Marktbereich bedient die Consumer Products Division von Digital Research mit Personal-BASIC, Personal-CP/M und einer eigenen LOGO-Version. Wie CP/M-86 ist auch Personal-CP/M auf ROM gespeichert und soll laut Vertrag mit Zilog bald auch auf einem Z80-Chip erhältlich sein. Digital Research nennt das „Microware" und ist davon überzeugt, die Vielzahl alternder Standard-CP/M-Programme dadurch zu aktivieren bzw. ihre Lebensdauer zu verlängern, daß man sie preiswert genug macht, um den Heimcomputer-Anwender zum Kauf zu motivieren.

Weitere vielversprechende Entwicklungen sind VIP und GSX. VIP ist ein preiswertes „Gerüst", das es Programmautoren möglich macht, unabhängig von den verwendeten Anwendungsprogrammen eine einheitliche Schnittstelle anzusteuern. Mehrere Applikationen nutzen dieselben Datas, und diese können vom einen zum anderen übertragen werden. Unter diesem Gesichtspunkt ist VIP Apples Lisa- und Macintosh-Technik sehr ähnlich, benötigt aber viel weniger Speicherplatz.

Was CP/M für den Diskettenbetrieb ist, ist GSX im Grafikbereich. Das Programm nutzt eine standardisierte Reihe von Grafikfunktionen, die auf verschiedenen Systemen läuft. Ein GSX-Programm kann auf einem Farbbildschirm oder einem Schwarzweißschirm laufen, auf einem Matrix-Drucker oder einem Plotter, ohne daß es einer Veränderung bedarf.

Digital Research hat sich als eines der wichtigsten Software-Häuser im Microcomputergeschäft etabliert. Doch man ruht sich nicht auf seinen Lorbeeren aus. Nach Entwicklung der Programme GSX, VIP und DR LOGO ist die Firma nun daran interessiert, wie Microsoft im Bereich der Anwendungsprogramme tätig zu werden. In Anbetracht des CP/M-Erfolges hat die Gesellschaft beste Zukunftschancen.

Kalkulationen

Eines der ersten Programme, das die Möglichkeiten der Microcomputer voll zur Geltung brachte, war die Tabellenkalkulation. Da diese Systeme ursprünglich für kommerzielle Zwecke entwickelt wurden, sind sie im Bereich der Heimcomputer nie recht akzeptiert worden. Kalkulationsprogramme können aber nicht nur Informationen sammeln und darstellen, sondern lassen sich auch als „Ideengenerator" verwenden.

Wie Textsysteme und Datenbanken haben auch Kalkualtionssysteme Fähigkeiten, die viele Anwender nicht nutzen. Die meisten Besitzer von Heimcomputern finden, die Tabellenkalkulation sei langweilig und ohne praktischen Nutzen, da sie mit dem kommerziellen Einsatz nichts anfangen können. Abgeschreckt vom negativen Image einer Buchhaltungsmaschinerie unterschätzen sie dabei die Möglichkeiten finanzieller Planung. Sie übersehen, daß Kalkulationssysteme für Planungen die gleiche Bedeutung haben wie Textverarbeitungssysteme für den Entwurf von Schriftstücken.

Kalkulationssysteme sind eigentlich Synthesen aus Textsystem und Taschenrechner. Da ihre Datenfelder in Zeilen und Spalten angeordnet sind, werden sie auch als Tabellenkalkulationsprogramme bezeichnet. Ähnlich wie bei Tabellierpapier lassen sich die Felder auf mehrere Arten beschriften: Außer Text und numerischen Daten können sie mathematische Formeln enthalten, die die Inhalte verschiedener Zellen miteinander verknüpfen. Mit diesen Formeln verwandelt der Anwender das Kalkulationssystem in ein individuelles Programm, in das nur die entsprechenden Daten eingesetzt werden müssen. Bei der Eingabe von Daten berechnen automatisch alle angesprochenen Formeln ihren Zelleninhalt neu, so daß ständig ein aktueller Stand angezeigt wird.

Kalkulationssysteme können aber nicht nur Berechnungen anstellen, die sonst zu umständlich wären, sondern auch eine Reihe anderer Aufgaben erfüllen. Beispiele dafür sind der Druck von Inventarlisten, die Analyse von Sportergebnissen, Tabellenentwürfe, die Erstellung von Stichwortlisten für die Beleuchtungs- und Klangeffekte eines Theaters, die Berechnung des Lohnsteuerjahresausgleichs, die Untersuchung, ob die Miete oder der Kauf eines Fernsehers günstiger ist und vieles mehr. Sollten all diese Aufgaben in BASIC programmiert werden, dann würden Stunden vergehen, bis die vielen PRINT TABs, PRINT ATs und INPUTs fehlerfrei eingegeben sind. In einem Kalkulationssystem entsteht das Format jedoch fast automatisch, wenn die Beziehungen zwischen den einzelnen Variablen festgelegt werden, wobei die bildschirmgesteuerte Eingabe so unkompliziert ist wie der Eintrag von Daten auf Tabellierpapier.

Eine ganze Reihe von Spezialbefehlen vereinfacht den Aufbau der Bildschirmanzeige: Felder lassen sich einzeln oder blockweise kopieren, bewegen und löschen, ganze Zeilen oder Spalten können eingesetzt oder gelöscht werden, und das Format von Zellenblöcken (Zellengröße, Position des Dezimalpunktes) läßt sich einheitlich gestalten.

Auch die Kalkulationsfunktionen sind einfach zu bedienen. Ein einziger Befehl berechnet den Durchschnittswert einer Zeile oder Spalte, stellt fest, welche Tabelleneinträge ungleich Null sind, kalkuliert die Summe einer Matrix oder zeigt das Minimum und das Maximum einer Zahlenkolonne an. All diese Funktionen lassen sich mit den herkömmlichen mathematischen Zeichen wie "+", "/", SQR und ABS kombinieren. Die Zahl der Möglichkeiten hängt hauptsächlich von der Größe des Arbeitsspeichers und der Qualität des Programms ab, wobei einige Kalkulationssysteme nicht alle dieser Funktionen unterstützen.

Problemlose Handhabung

Einer der brauchbarsten Befehle einer Tabellenkalkulation ist REPLICATE. Er überträgt den Wert oder die Formel eines Feldes in andere Felder. Damit läßt sich eine Tabelle mit identischen Datenfeldern (zum Beispiel eine Liste der monatlichen Hypothekenzinsen) durch die Eingabe weniger Befehle zusammenstellen. Da sich komplizierte mathematische Ausdrücke mit einem Kalkulationssystem einfacher darstellen lassen als in BASIC, kann die Arbeit damit schon nach kurzer Zeit so vertraut werden wie die mit mathematischen BASIC-Befehlen.

Einmal angelegte Tabellen können auf Cassette oder Diskette gespeichert werden. Viele Programmversionen haben außerdem die Möglichkeit, Text und Daten in ein Dateiformat zu übertragen, das sich zur Weiterbearbeitung durch Textsysteme oder Datenbanken eignet. Die Ergebnisse von Berechnungen und Projektionen lassen sich so als Block in einen Text

In Reih und Glied

Formatieren
Mit dem Befehl FORMAT wurden Spaltenbreiten, Textstand und Zahlenanzeige festgelegt.

Kopieren
Jedes Datenfeld kann mit dem Befehl COPY in jeden anderen Bereich der Tabelle kopiert werden.

Multiplikationsfaktor
Um einen Vergleichswert zu erhalten, wird die Note mit diesem Faktor multipliziert.

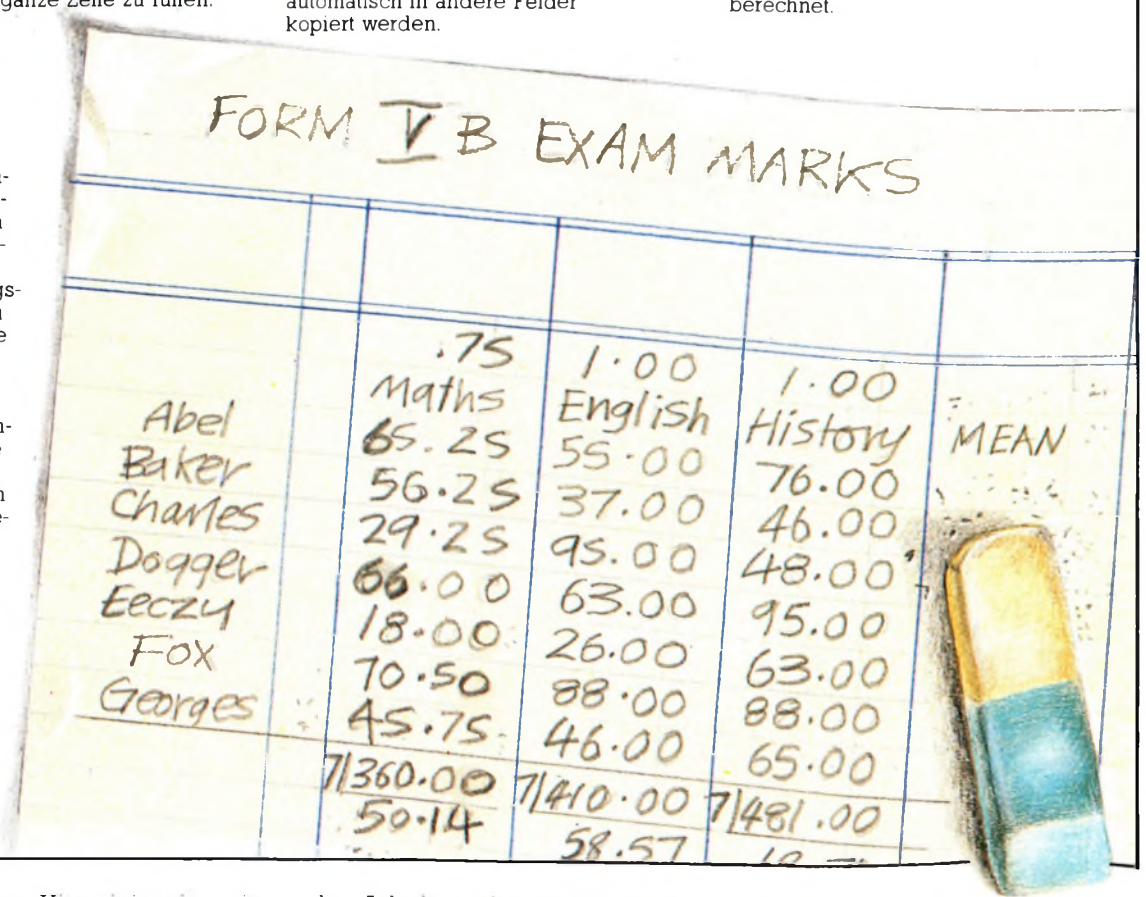
```
    |  C  |  D  |  E  |  F  |  G  |  H  |  I  |  J  |  K  |  L  |  M  |
 1| MARKS ADJUSTMENT EXAMPLE
 2|*************************************************************************
 3|                  ACTUAL   MARKS      *        SCALED   MARKS
 4|*************************************************************************
 5|                                      *      .75     .86     .73
 6|          Maths     Eng     Hist     MEAN  *  Maths     Eng     Hist     MEAN
 7|Abel    : 87.00    55.00    76.00    72.67  *  65.25    47.30    55.48    56.01
 8|Baker   : 75.00    37.00    46.00    52.67  *  56.25    31.82    33.58    40.55
 9|Charles: 39.00    95.00    48.00    60.67  *  29.25    81.70    35.04    48.66
10|Dogger  : 88.00    63.00    95.00    82.00  *  66.00    54.18    69.35    63.18
11|Eezy    : 24.00    26.00    63.00    37.67  *  18.00    22.36    45.99    28.78
12|Fox     : 94.00    88.00    88.00    90.00  *  70.50    75.68    64.24    70.14
13|George  : 61.00    46.00    65.00    57.33  *  45.75    38.56    47.45    44.25
14|          ========================          ========================
15|  MEAN  : 66.86    58.57    68.71    64.71  *  50.14    50.37    50.16    50.23
16|                                      *
17|*************************************************************************
18|
```

Textwiederholung
In Verbindung mit dem Befehl REPEAT TEXT genügt hier die Eingabe eines einzigen Sterns, um die ganze Zeile zu füllen.

Automatische Berechnung
Ist die Formel einmal in ein Feld eingegeben, kann sie mit dem Befehl REPLICATE automatisch in andere Felder kopiert werden.

Durchschnitt
Wird von dem Befehl AVERAGE (Feld#1:Feld#2) berechnet.

Um die Leistungen seiner Schüler in verschiedenen Fächern vergleichen zu können, suchte ein englischer Lehrer nach Maßstäben, die eine einheitliche Beurteilung zulassen. Er experimentierte für jedes Fach mit verschiedenen Multiplikationsfaktoren und mußte dabei die Leistungswerte immer wieder neu berechnen. Diese lästige und fehleranfällige Arbeit führt ein Kalkulationssystem in wenigen Minuten aus. In der Computertabelle werden alle Werte (außer den Noten der Schüler) automatisch berechnet. Bei der Änderung eines Multiplikationsfaktors entstehen in Sekunden die neuen Vergleichswerte dieses Faches.



oder eine Datei einfügen. Hiermit ist ein weiterer wichtiger Schritt in Richtung auf eine integrierte Datenverarbeitung getan. Diese Möglichkeit gibt es jedoch nur bei den teureren Programmpaketen.

Kalkulationssysteme finden ihre Grenzen in der Kreativität des Anwenders und der Größe des Arbeitsspeichers. Da die Programme normalerweise viel Speicherplatz belegen, kann der Einsatz großer Tabellen und komplexer Berechnungsvorgänge den vorhandenen Speicher schnell ausfüllen. Zu komplizierte mathematische Formeln verlangsamen die Arbeitsgeschwindigkeit der Programme.

Branchenwechsel

Die Entwicklung der Tandy Corporation begann bereits vor über fünfzig Jahren – ein weiter Weg vom Lederlieferanten zum größten Computerhändler der Welt.

Die Tandy Corporation hat mit ihren Filialunternehmen Tandy und American Radio Shack eine Ladenkette von fast 400 Computerzentren und über 5500 Niederlassungen in 76 Ländern. Das Unternehmen besitzt 29 Fabriken, in denen die Produkte hergestellt werden, die unter den Markennamen Tandy und Radio Shack im Angebot sind.

Tandy wurde jedoch nicht als Elektronikfachgeschäft gegründet. Norton Hinckley und David Tandy begannen 1927 mit der Hinckley-Tandy Leather Company, einer Firma, die als Lederlieferant für Schuster in Beaumont, Texas, tätig war. Der erste geschäftliche Schritt in Richtung Elektronik-Gigant erfolgte 1963, als David Tandys Sohn Charles sich entschloß, das Unternehmen zu erweitern. Er erwarb Anteile an einer in Boston ansässigen Firma namens Radio Shack, die beträchtliche Probleme hatte.

Übernahme von Radio Shack

Seit Anfang der zwanziger Jahre war dieses Unternehmen auf den Vertrieb von elektronischen Bauteilen spezialisiert. Obwohl das Geschäft überwiegend auf dem Versandwege abgewickelt wurde und im Raum Boston insgesamt neun Niederlassungen existierten, machte Radio Shack große Verluste. 1967 war es Charles Tandy bereits gelungen, den Verlust von vier Millionen Dollar in einen Gewinn von 20 Millionen Dollar umzuwandeln.

Der nächste entscheidende Schritt erfolgte 1970 mit der Übernahme einer Ladenkette namens „Leonards", die Produkte der Unterhaltungselektronik verkaufte und Tandy den Einstieg in diesen Marktbereich ermöglichte. Im Tandy-Katalog von 1984 sind 2625 Artikel aufgeführt, die nichts mit Computern zu tun haben, vom Widerstand über HiFi-Bausteine bis hin zu Synthesizern. Dazu kommen 396 Computer-Artikel.

Die erste englische Tandy-Niederlassung wurde 1973 eröffnet. Das Haus erarbeitete sich rasch einen guten Ruf als Elektronikspezialist. 1978, als der „TRS-80 Modell I" eingeführt wurde, verfügte Tandy in England bereits über 120 Niederlassungen. 1983 war die Zahl der Filialen bereits auf 227 gewachsen.

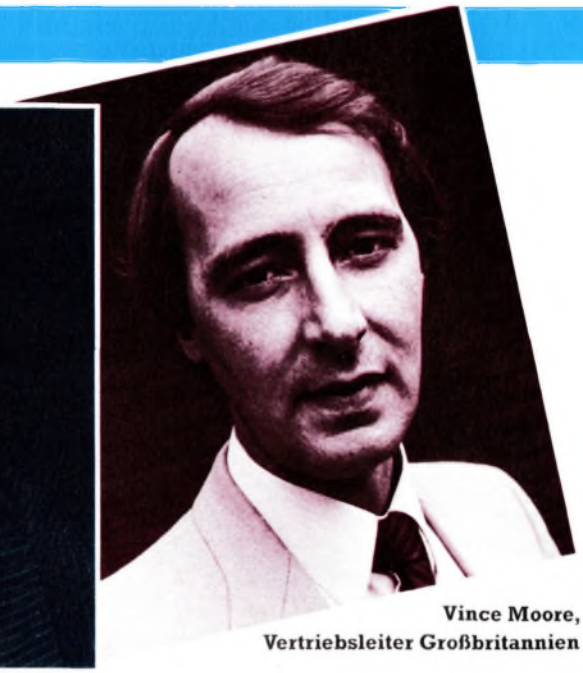
Mit dem „Modell I" verschaffte sich Tandy den Ruf eines führenden Computerherstellers. Dabei handelt es sich um einen Einplatinenrechner mit einem Z80-Microprozessor, vier K RAM-Speicher und einem Schwarzweißbildschirm mit niedrig auflösender Grafik.

Der Name Tandy ist längst zum Markenzeichen geworden. Damit sind drei Qualitätsbegriffe verbunden: Angebot, Service und Beratung. Unter dem Namen Radio Shack ist das Unternehmen auch in den USA bekannt.

Seitdem ist die Tandy-Computerproduktlinie ständig aktualisiert worden, wenngleich es nicht gelang, mit den Nachfolgemodellen an den Erfolg des „Modell I" anzuschließen. Das Unternehmen bietet inzwischen auch Geschäftscomputer an, so beispielsweise „Modell II" sowie „Modell 12" und „Modell 16". Bei den beiden letzteren handelt es sich um 16-Bit-Rechner. Das gilt auch für das neue, IBM PC-kompatible „Modell 2000". Inzwischen bietet Tandy das „Modell 4" an (und dazu eine tragbare Version, das „Modell 4P"). Beide sind besonders für den Geschäftsbereich geeignet und – das ist bei Microcomputern ungewöhnlich – kompatibel zu den Modellen I und III.

Mit dem „Tandy Color Computer", basierend auf dem 6809-Microprozessor, versuchte das Unternehmen, seine Position im Markt wieder-

John Sayers, Marketing-Direktor Großbritannien

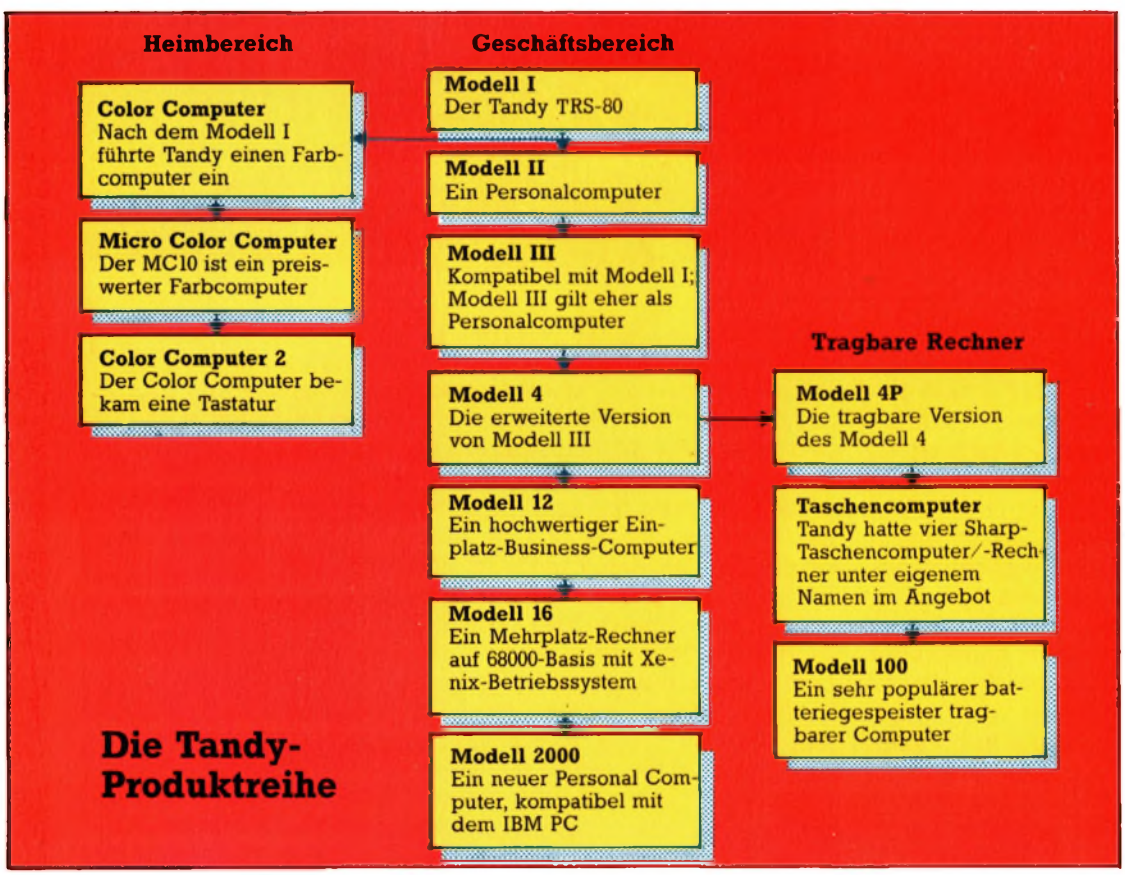


Vince Moore, Vertriebsleiter Großbritannien

zuerlangen. Dieser Rechner ist in vielen Teilen dem „Dragon" ähnlich. Eine weiterentwickelte, zugleich verkleinerte Version mit der Bezeichnung „Micro Color Computer" ist Indiz dafür, daß Tandy in diesem Bereich weiter arbeitet.

Die vielleicht interessanteste Produktlinie ist die der tragbaren Computer. Tandy verkaufte zunächst Taschencomputer, die den Sharp-Rechnern ähnlich waren. Das neue „Modell 100" resultierte aus der Zusammenarbeit mit dem japanischen Anbieter Kyocera und dem amerikanischen Software-Haus Microsoft. Das Ergebnis: Ein buchgroßer, batteriegespeister tragbarer Rechner mit integriertem BASIC, Kommunikations-Software und „Tagebuch".

Viele Computerunternehmen mußten 1984 große Verluste hinnehmen. Tandy dagegen arbeitet außerordentlich erfolgreich und bringt ständig neue Produkte heraus, die sowohl in Fort Worth, Texas, dem neuen Sitz der Gesellschaft, als auch in der TC Electronics Corporation, der japanischen Niederlassung von Tandy in Tokio, entwickelt werden. Die Tandy Corporation scheint auch langfristig einen festen Platz im Microcomputer-Geschäft beanspruchen zu können.



Die Übersicht zeigt die drei Geschäftsbereiche der Tandy-Rechner und ihre Entwicklung seit Beginn der Produktion.

Schreiben für den Bildschirm

Viele Heimcomputer-Programmierer träumen davon, einen Bestseller zu schreiben. Doch sie sind sich selten dessen bewußt, welcher Aufwand damit verbunden ist.

Natürlich bedeutet der Besitz einer teuren Ausstattung noch lange nicht, daß auch die Programme erfolgreich sind. Es gibt Amateure, die mit ihrer auf dem kleinen Spectrum zu Hause entwickelten Software ein Vermögen verdient haben. Und doch sind diese „Heimhexer" auf dem Computer vom „Aussterben" bedroht, wenn man die Entwicklung der großen Software-Häuser in den vergangenen Jahren betrachtet. Dank ihrer Großrechner und ausgetüftelten Programmierhilfen sind sie den Amateur-Programmierern gegenüber im Vorteil.

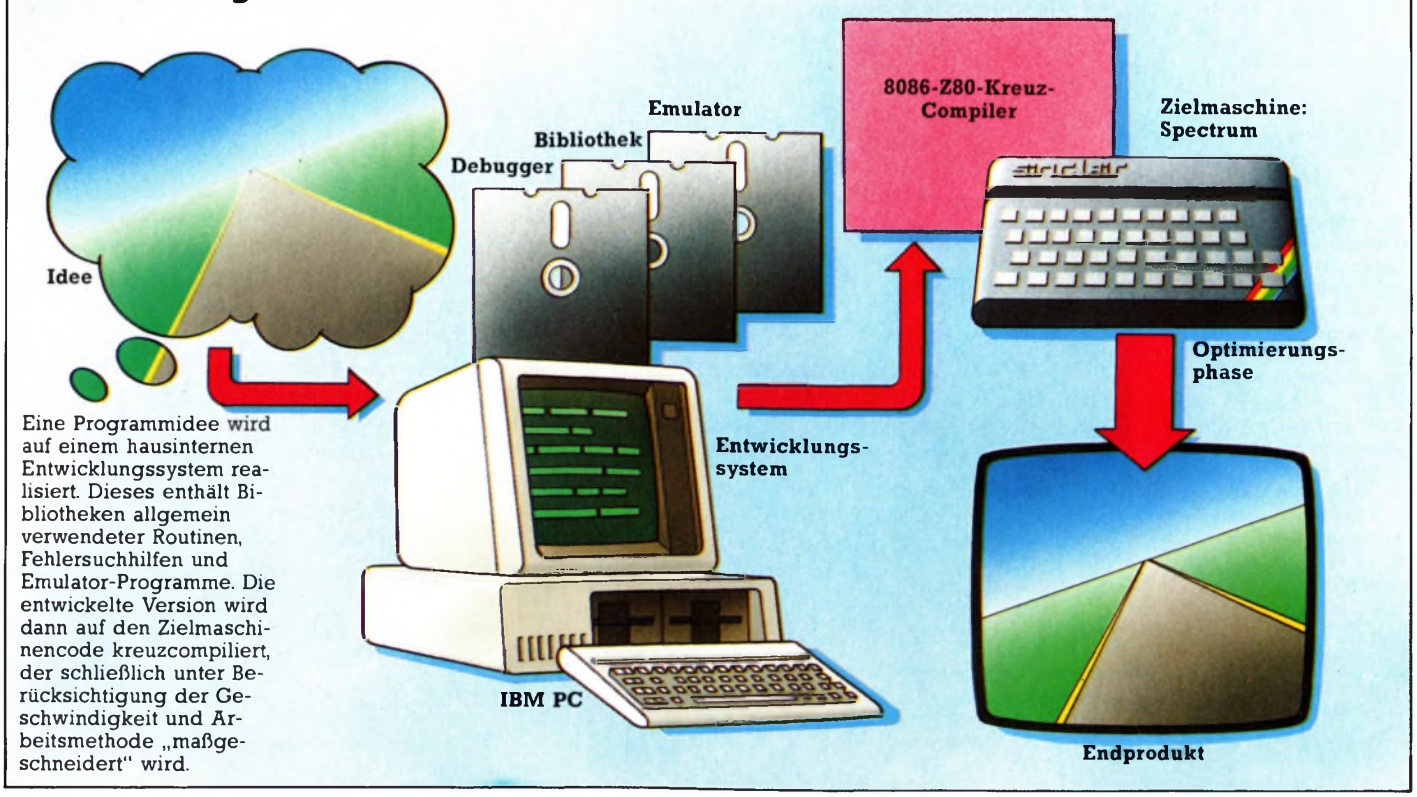
Einer der wichtigsten Punkte bei der Software-Entwicklung für den Heimcomputermarkt ist die Verarbeitungsgeschwindigkeit. Daraus folgt, daß die Programme in Maschinensprache geschrieben sein müssen. Doch die Maschinensprache zu beherrschen, ist nicht leicht. Programmierer, die damit arbeiten, benötigen besondere Software, um ihre Programme schreiben zu können. Zumindest ist ein Assembler-Programm zur Übersetzung des Quellcodes in den Objektcode erforderlich, den die Maschine versteht. Die Aufgabe ist besonders schwer, wenn das Programm umfangreich ist.

Heimcomputer genügen nicht

Die Qualität der für Heimcomputer zur Verfügung stehenden Assembler-Programme ist recht dürftig. Selbst die einfachsten Programmpakete dieser Art belegen viel Speicherplatz, womit auch die Programmlänge begrenzt wird. Viele Heimcomputer haben konstruktionsbedingte Nachteile, wie schlechte Tastaturen oder mangelhafte Bildschirmdarstellung, die das Erarbeiten von Programmen zur Qual machen.

Aus diesen Gründen schreiben die meisten kommerziellen Unternehmen die Programme nicht auf den Rechnern, für die sie gedacht

Entwicklungsabläufe



Eine Programmidee wird auf einem hausinternen Entwicklungssystem realisiert. Dieses enthält Bibliotheken allgemein verwendeter Routinen, Fehlersuchhilfen und Emulator-Programme. Die entwickelte Version wird dann auf den Zielmaschinencode kreuzcompiliert, der schließlich unter Berücksichtigung der Geschwindigkeit und Arbeitsmethode „maßgeschneidert" wird.

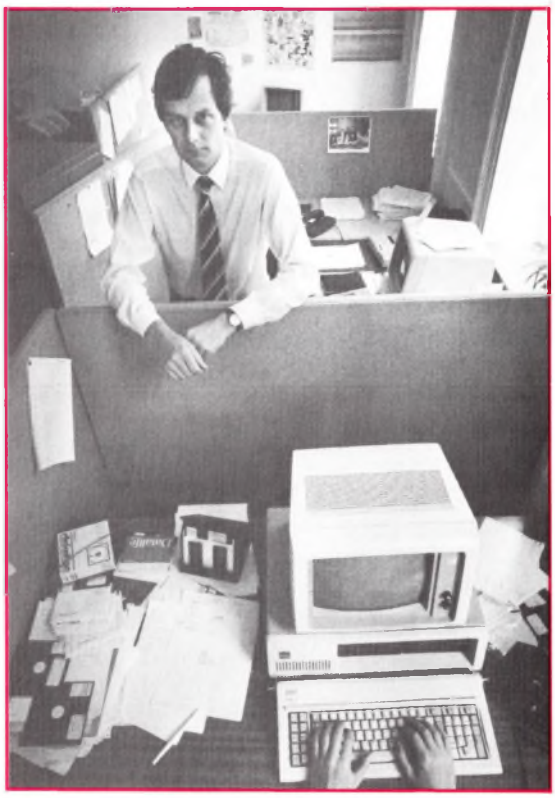
Auf strategische Spiele wie Schach spezialisiert, verwendet IS bei der Software-Entwicklung IBM- und Apple-Computer, die mit selbstentwickelten Schnittstellen ausgestattet sind. Die Aufteilung der Programme in maschinenabhängige und universell einsetzbare Segmente vereinfacht es, gleichzeitig mehrere Computer und spezielle Schachcomputer mit Programmen zu versorgen.

sind (die sogenannte „Zielmaschine"), sondern benutzen zur Programmentwicklung professionelle Systeme mit spezieller Software (man nennt diese „Entwicklungssysteme"). Programmierer, die diese Systeme benutzen, schreiben in Sprachen wie PASCAL oder C. Dabei kommen Sprachversionen zur Anwendung, die als Kreuz-Compiler oder Kreuz-Assembler bezeichnet werden. Das bedeutet: Man kann auf einem Rechner arbeiten, der einen 8086-Prozessor hat, und die erzeugten Programme laufen auf einem Rechner, der mit einem Z80 ausgestattet ist.

Natürlich hat ein solches Entwicklungssystem erhebliche Vorteile gegenüber einem Heimcomputer. Ein Assembler auf Diskette oder ergänzende RAM-Kapazitäten zur Speicherung größerer Programme erleichtern die Arbeit. Debugging-Routinen können in die Rohversion des Programms geladen werden, ohne daß man Sorge haben muß, daß der Speicherplatz nicht reichen könnte. Dazu kommen die besseren Arbeitsmöglichkeiten auf einem Rechner mit guter Tastatur, brillanter Bildschirmdarstellung und Diskettenstationen.

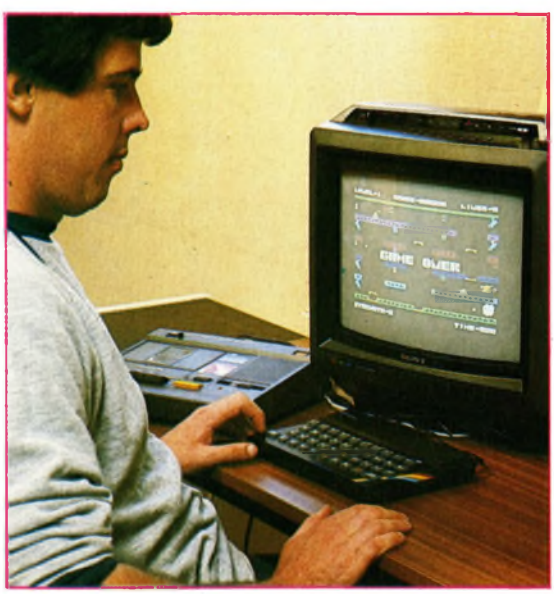
Eine der Firmen, die sich dieser Programmentwicklungstechnik bedient, ist Intelligent Software (IS). Sie wurde 1981 von erfahrenen Programmierern gegründet, so dem Schachspezialisten David Levy und Robert Madges von ANT Microware. Das Unternehmen ist auf strategische Spiele spezialisiert, die zumeist Auftragsproduktionen bekannter Heimcomputerhersteller sind. Das Haus entwickelt außerdem Software für Schachcomputer.

Neben den Computern, die für die Entwicklung eingesetzt werden, benutzt IS IBM PCs und Apple-Rechner mit speziell entwickelten Schnittstellen, die die Übertragung von Codes zwischen den verwendeten Rechnern ermöglichen. Das Unternehmen wird häufig damit beauftragt, Programme zu übertragen – etwa ein vorhandenes Schachprogramm auf einen anderen Computer umzuschreiben. Folglich mußten die Programmierer lernen, den Code in segmentierter Form zu schreiben. Ein Teil dieser Segmentierung hat sich als gerade dann besonders nützlich erwiesen, wenn der Code vom einen Prozessor auf den anderen übertragen wird. Und zwar ist dies die Teilung des Programms in einen Spielcode und einen Eingabe/Ausgabe-Code. Der Input/Output-Code eines neuen Rechners hat häufig einen anderen Port oder muß anders adressiert werden oder ist vielleicht sogar von der Strategie her anders (beispielsweise muß eine Unterbrechung erfolgen).



Da IS die Programmierer so wenig wie möglich einschränken will, gibt es für die Erstellung der Codes nur wenige „Hausregeln". Ein wichtiger Punkt, auf dem man besteht, ist jedoch, daß der Quellcode reichlich Anmerkungen enthalten soll, damit man immer weiß, was welche Routine tut.

Die meisten Vision-Programme werden von zu Hause arbeitenden Programmierern auf den Zielmaschinen entwickelt. Nachdem Spielkonzept und Bilschirmdarstellung festgelegt wurden, erfolgt die Entwicklung der einzelnen Routinen in herkömmlicher Assembler-Sprache (Z80 oder 6502) unter Verwendung von Assemblern wie dem HiSoft DevPak auf dem Spectrum.



Gut ausgerüstet: Psion

Das Softwarehaus Psion benutzt Computer, die leistungsfähiger sind als der IBM PC. Psion hat unter den vielen britischen Softwarehäusern, die Programme für Heimcomputer entwickeln, eine Sonderstellung, da die meiste Arbeit auf Minicomputern ausgeführt wird.

Psion begann als Unternehmen, das Software für den ZX 81 schrieb – und verwendete den ZX 81 auch dafür. Als es an die Programmierung von „Horizons" ging, das mit jedem Spectrum geliefert wird, erwarb Psion einen TRS 80 mit Diskettenstation, einen Rechner,

der den gleichen Z80-Prozessor hat, und baute ein spezielles Interface für die beiden Maschinen. Doch im August 1982 befand das Haus, daß es unmöglich sei, jedesmal dann, wenn ein neuer Heimcomputer auf den Markt kommt, ein völlig neues und anderes Entwicklungssystem zu schaffen. Deshalb investierte man in leistungsfähigere Hardware mit hoher Rechengeschwindigkeit. Diese Hardware wird allen Problemstellungen gerecht, die künftig entwickelte Computer mit sich bringen. Man entschied sich für zwei Vax 750 und das Betriebssystem VMS von DEC.

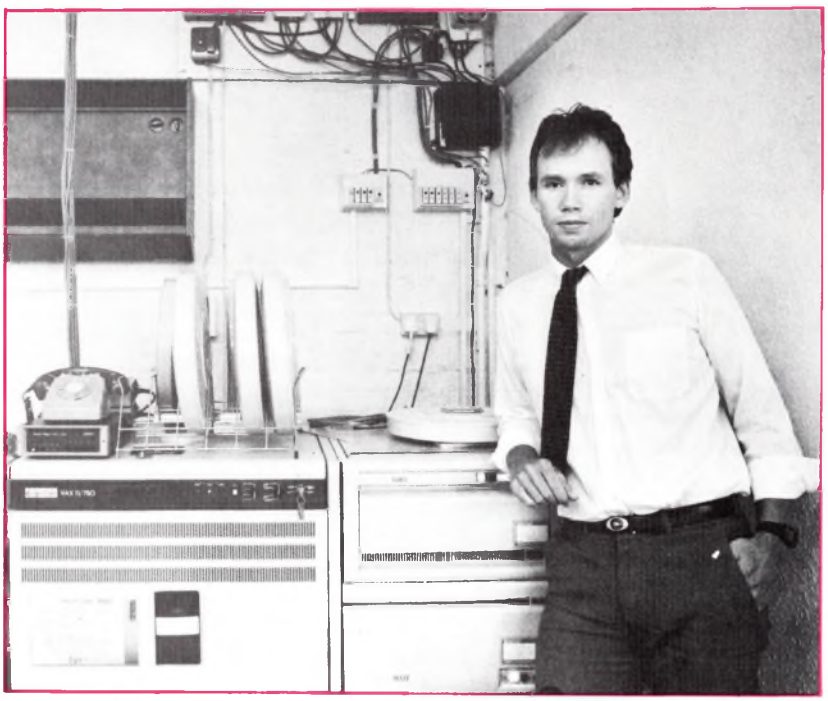
Vorteile der Vax

Die Vax 750 brachten Psion zwei Vorteile: die Qualität der Software, die DEC bietet, verbunden mit der Möglichkeit, speziell entwickelte Software-Hilfen zu kreieren, einerseits und die Leistungsfähigkeit des Betriebssystems und der Hardware-Kombination andererseits. Es gibt genügend Speicherplatz für eine ganze Software-Hilfs-Bibliothek, wie Compiler, Sammlungen üblicher Subroutinen und Fehlerbeseitigungs-Programme, die sechzehn bis zwanzig Programmierer gemeinsam nutzen können, die gleichzeitig am System [illegible].

Die neueren Vax-Rechner erlauben Autorenteams gemeinsames Arbeiten, Teilung von Projektbibliotheken, aus denen fast augenblicklich Module abgerufen werden können, und die vorhandenen Bibliotheken können sogar gleichzeitig von an unterschiedlichen Projekten arbeitenden Teams genutzt werden. Darin besteht der große Vorteil dieses sogenannten „Timesharing"-Systems. Dazu kommt, daß darauf die gesamte Verwaltungsarbeit erledigt werden kann, ohne daß die Programmierer ihre Arbeit unterbrechen müssen. Psion beabsichtigt, eine dritte Vax zu installieren, die ausschließlich für die Abwicklung administrativer Arbeiten verwendet werden soll.

Selbst wenn man es sich leisten könnte, so würde man durch den Erwerb einer Vax nicht automatisch zum Konkurrenten von Psion werden. Nur ein Bruchteil des verwendeten Materials wurde Psion von DEC als Fertigprogramm geliefert. Harte Arbeit war erforderlich, um einfache Aufgaben wirtschaftlich durchführen zu können. In C geschriebene Software-Hilfen und Utilities hat Psion selbst dem Grundsystem hinzugefügt.

Psion verwendet C, eine „Mittelhoch"-Sprache, mit der sehr kompakte und schnelle Objekt-Codes für 16-Bit-Chips wie den 8086 geschrieben werden können. Die Firma mußte jedoch eigene Techniken entwickeln, um Programme für Zielmaschinen wie den Spectrum erstellen zu können. Natürlich gibt das Unternehmen seine Geheimnisse nicht preis. Doch man weiß, daß Psion C benutzte, um einen eigenen Compiler zu schreiben, der einfach „Tischsprache" genannt wird.



Psion erwarb 1982 zwei Vax-750-Minicomputer, um eine Basis für die Softwareentwicklung zu schaffen. An jedem Rechner können bis zu 20 Programmierer gleichzeitig arbeiten und sämtliche Kreuz-Compiler nutzen, zudem Software-Bibliotheken und Fehlersuchprogramme, um so Programme erstellen und übersetzen zu können.

Es gibt eine Grundregel, die besagt, daß die Systemwartung und die Erstellung von Programmierhilfen wie der „Tischsprache" etwa 30 Prozent aller Programmierarbeit in Anspruch nehmen. Psion sieht diesen Zeitaufwand als gerechtfertigt und die Investition als Anlage an. Ein im eigenen Haus entwickelter Quellcode bietet zahlreiche Vorteile: Man kann ihn zerlegen und verbessern oder auf eine Art umwandeln, die bei der Verwendung fertig konfektionierter Software ganz unmöglich ist. Bei Software, die von Dritten erstellt wurde, ist es schwer, wenn nicht unmöglich, einmal vorhandene Fehler auszumerzen.

Die von Psion erworbene Software beinhaltet Programme, die exakte Simulationen populärer Microprozessoren wie des Z80 oder des 6502 sind. So lassen sich die DEC-Vax-Computer programmieren, als handelte es sich bei ihnen um einen Commodore 64 oder einen Spectrum. Trotz der Kapazität der Vax-Rechner laufen die Simulatoren nur mit einem Bruchteil der Geschwindigkeit der Zielmaschinen. Vorteil dabei ist, daß der Programmierer jederzeit auf den Inhalt jedes Registers im Microprozessor Zugriff hat, das Programm also in jedem Stadium genau verfolgen kann.

Die neueren Entwicklungen Psions sind vier Standard-„Business"-Programme für den Sinclair QL, die zum Lieferumfang des Rechners gehören. Die Motorola-68000-Chip-Familie, von denen einer die Zentraleinheit des QL ist, wurde auf Hochsprachenebene entwickelt. C-Programme kompilieren so gut auf diese Chips, daß sich das Schreiben in Maschinensprache erübrigt. Würden alle Heimcomputer dem QL-Vorbild folgen, könnte C die Maschinensprache völlig ersetzen. Softwarehäuser wie Psion könnten dann die Übertragung per Hand auf alle Zeiten vergessen.

Imagepflege

Melbourne House ist für Abenteuerspiele bestens bekannt. Darunter sind solche Hits wie „The Hobbit" und „Mugsy", die sich durch hohen Grafikstandard und gute Geschichten auszeichnen. Doch das Unternehmen produziert auch Computerbücher.

Melbourne House wurde 1977 von dem Australier Alfred Milgrom gegründet. Die Einführung des Sinclair ZX 80 brachte Milgrom auf die Idee, mit dem ertragreichen Verlegen von Heimcomputerbüchern zu beginnen. 1980 veröffentlichte Melbourne House „30 Programme für den ZX 80". Der Erfolg dieses Buches wurde mit einer ganzen Serie von Büchern für den Sinclair-Rechner fortgesetzt.

Im Jahr darauf kam der ZX 81 auf den Markt. Die Nachfrage nach Büchern und Programmen für den ZX 80 ließ daher nach. Die Gesellschaft überlebte diese Phase nur, weil die Verkäufe in den USA trotzdem gut waren. Melbourne House lernte aus dieser Lektion, und man erkannte die Notwendigkeit, sich auch mit anderen Computern zu befassen.

Aufgrund des Erfolges des Sinclair Spectrum produzierte Melbourne House Spielprogramme, die die Grafik- und Soundmöglichkeiten dieses Rechners voll ausnutzten. Das Spielhallenspiel „Penetrator" verkaufte sich ausgezeichnet, doch das erfolgreichste Programm des Unternehmens war „The Hobbit", ein Grafikabenteuer, das auf Tolkiens gleichnamigem Roman beruht. Es wurde mit dem „Goldenen Joystick" als bestes strategisches Spiel des Jahres ausgezeichnet. Die Programmcassette wurde als Paket vermarktet, in dem auch ein Exemplar von Tolkiens Buch war. Dies war eine Auflage der Nachlaßverwalter von Tolkien und hatte zur Folge, daß The Hobbit dreimal so teuer war wie andere für den Spectrum erhältliche Software. Trotz des Preises lief der Verkauf gut. Jetzt ist das Spiel auch für andere Heimcomputer erhältlich.

Die hauseigenen Programmierer der Firma arbeiten in Melbourne, Australien. Jeweils vier Leute befassen sich bei der Programmentwicklung mit einem Spielaspekt. So braucht man für die Fertigstellung viel Zeit, doch der Aufwand scheint durch Erfolg und Umsätze gerechtfertigt. Die Loyalität der Kunden will das Unternehmen nutzen, um so mehr Bücher verkaufen zu können. Paula Byrne, Pressesprecherin des Hauses, sagt dazu: „Wenn Leute ein Buch kaufen wollen, wissen sie nicht, für was sie sich entscheiden sollen. Vielleicht kaufen sie am Ende etwas, das ihnen nicht gefällt. Und das entmutigt sie, weitere Bücher zu kaufen." Dieser Verwirrung will Melbourne House dadurch entgegenwirken, daß man die Bücher kennzeichnet als geeignet für Einsteiger, Fortgeschrittene oder erfahrene Leser. Durch das übereinstimmende Erscheinungsbild von Büchern und Software soll der Konsument die hochwertigen Bücher und die gute Software als gleichwertig ansehen. Ferner ist jedem Exemplar eine Karte beigefügt, mit der Leser aufgefordert werden, ihre Meinung zur Qualität des Produktes mitzuteilen.

Ein neueres Spiel des Hauses, „Mugsy", wird als der „beste interaktive Computer-Comic-Strip der Welt" bezeichnet. Dabei übernimmt der Spieler die Rolle eines Bandenchefs im Chicago der zwanziger Jahre. Die Grafiken sind ausgezeichnet und sehr detailliert. Melbourne Hose arbeitet außerdem an einem Spiel in Hobbit-Art, betitelt „Sherlock Holmes", das jetzt fertiggestellt sein müßte. Es soll ebenso innovativ wie The Hobbit sein. Man benötigte für die Entwicklung fünfzehn Monate. Über den Spielinhalt ist bisher kaum etwas bekannt. Man weiß lediglich, daß gute Kenntnisse „viktorianischen Transportwesens" erforderlich sind, um erfolgreich zu sein.

Alfred Milgrom, Direktor und Verleger, Melbourne House

Philip Mitchell, Autor von The Hobbit und Sherlock Holmes.

Melbourne House hat mit The Hobbit ein witziges und aufregendes Spiel geschaffen. Zum Lieferumfang gehört auch ein Exemplar von Tolkiens Meisterwerk – eine ausgezeichnete Marketingidee.

Ein Teil der Melbourne-Mannschaft

Gebrüder Casio

Casio ist als Hersteller von Uhren, Taschenrechnern und Musikinstrumenten bekannt. In den vergangenen Jahren brachte das Unternehmen zahlreiche Hand-held- und Taschencomputer heraus, um so auf diesen speziellen Markt zu gelangen.

Casio behauptet, den Taschenrechner-Markt weltweit mit 50 Prozent zu beherrschen, beschäftigt aber nur etwa 3300 Angestellte. 1983 machte Casio einen Gesamtumsatz von 29 Millionen US-Dollar – für einen Elektronikhersteller dieser Größenordnung nicht besonders viel.

Casio wurde von den fünf Brüdern Kashio nach dem Zweiten Weltkrieg gegründet. Damals hieß die Firma Kashio Seisakujo und stellte Büromaterialien her. Anfang der fünfziger Jahre entwickelte man den 14-A-Relay-Calculator (Relaisrechner). Er war einer der ersten elektrischen Rechner, hatte die Größe eines Schreibtisches und wog 130 kg. 1957 erfolgte dann die Namensänderung in Casio Computer Company Ltd.

Mißerfolg des FX-702P

Im Jahre 1982 führte Casio den ersten Taschencomputer ein, als FX-702 P bezeichnet. Er war für wissenschaftliche Anwendungen konzipiert und verfügte über eine ungewöhnliche Tastatur-Anordnung: Die Tasten waren nämlich alphabetisch aufeinanderfolgend gesetzt und nicht im üblichen QWERTY- bzw. QWERTZ-Standard angelegt. Aufgrund der enttäuschenden Verkaufsergebnisse ersetzte man den Rechner durch das Modell FX-700 P, das mit QWERTY-Tastatur ausgestattet war.

Darauf folgte der PB-100, ein Taschencomputer für Geschäftsleute. Ähnlich gestaltet wie der FX-702 P, war er mit einer Flüssigkristall-Anzeige und numerischer Tastatur versehen. Dazu lieferte Casio eigene Cassettenrecorder, Drucker/Plotter und RAM-Erweiterungen für beide Modelle.

FX-750 P ist das jüngste Nachfolgemodell des PB-100. Dieser Computer verfügt über zwei „RAMcard"-Schächte, in die schmale Metallmodule (in Streichholzschachtelgröße) passen, auf denen bis zu vier KByte gespeichert werden können. Jede Karte enthält eine Drei-Volt-Batterie, so daß die Daten nicht verlorengehen, wenn die Karte aus dem Computer entfernt wird. Programme lassen sich auf diese Weise beliebig speichern oder laden.

Casio stellt außerdem den FP-200 her, einen Hand-held-Computer mit acht KByte RAM, der auf 32 KByte erweitert werden kann. Der FP-200 verfügt über eine LCD-Anzeige. Im Text-Mode werden acht Zeilen mit je 20 Zeichen dargestellt. Die Grafikauflösung beträgt 160 x 64 Punkte. Der neueste Rechner der Casio-Produktpalette ist der SL-800. Es handelt sich um einen preiswerten Taschenrechner von Größe und Gewicht einer Kreditkarte. Die extrem schlanke Form ist das Entwicklungsergebnis eines Herstellungsprozesses, den man „Filmen" nennt. Die Bauteile werden dabei auf einen speziell beschichteten Film gedruckt und nicht wie bei den herkömmlichen Verfahren auf eine Platine gelötet.

Casio vertreibt in vielen europäischen Ländern und dem Fernen Osten Großrechner und MSX-Computer. In England aber bietet das Unternehmen ausschließlich Taschenrechner und kleine Computer an. Auf die Frage, warum Casio nicht versucht, in den englischen Heimcomputer- und Geschäftscomputer-Markt zu gehen, verweist Tony Manton, Verkaufsleiter für Taschenrechner, auf die Gefahren, die dieser Industriebereich birgt: „Wir wollen langsam wachsen", sagt er. „Wir müssen die Leute erziehen und ihnen verdeutlichen, daß Taschen- und Hand-held-Computer mehr als nur Taschenrechner sind."

Casio Forschungs- und Entwicklungszentrum in Tokio

Tadao Kashio, Präsident von Casio, ist eines der fünf Familienmitglieder der Kashios, die das Unternehmen Casio führen.

Natürliche Auslese

Viele Millionen Microprozessoren sind weltweit in Gebrauch, bei Videorecordern und Microwellengeräten ebenso wie bei Computern. Dabei wird der Heimcomputermarkt im wesentlichen von nur zwei Prozessortypen beherrscht: dem Z80 und dem 6502.

Bei den meisten Heimcomputern beschränkt sich die Frage nach dem Prozessor darauf, ob nun ein 6502 oder ein Z80 eingebaut ist. Nur bei wenigen Rechnern kommen andere Prozessoren zum Einsatz; der Dragon arbeitet beispielsweise mit einem 6809.

Im Jahr 1972 wurde der Baustein-Hersteller Intel von Datapoint gebeten, einen Chip herzustellen, der die Vielzahl von TTL(Transistor-Transistor-Logik)-Schaltkreisen in den damaligen Computer-Terminals ersetzen sollte. Intel entwickelte daraufhin den Microprozessor 8008, der acht Bits parallel verarbeitete und der ideale „logische" Ersatz für die Gatterbausteine in den Datapoint-Terminals gewesen wäre – aber er arbeitete zu langsam und kam nicht wie geplant zum Einsatz. Entwickler und Bastler stellten jedoch schon bald fest, daß der 8008 eine vielseitige Rechner-Zentraleinheit ist. Das war die Geburtsstunde der preisgünstigen Schreibtisch-Rechner.

Die Grenzen des 8008 hinsichtlich Leistung und Geschwindigkeit waren aber doch so eng gesetzt, daß Intel wenig später als Weiterentwicklung den 8080 herausbringen mußte. Er dominierte sehr schnell den Markt.

Fast zeitgleich mit der Ankündigung des 8080 durch Intel stellte Motorola unter der Bezeichnung 6800 einen ganz anders konzipierten 8-Bit-Microprozessor vor. Sein Leistungsvermögen war durchaus ebenbürtig, und beide Prozessoren schienen gleich gut als Basis für Microcomputer geeignet.

1974 entwickelte Gary Killdall, der spätere Chef von Digital Research, bei Intel ein Disketten-Betriebssystem (DOS) namens CP/M. Dieses System ermöglichte den Einsatz der neuen Floppy-Laufwerke bei Rechnern mit 8080-Prozessor. Es wurde jedoch von Intel nicht übernommen, weil man vorrangig Großrechner im Auge hatte, für die ohnehin ausreichend Software vorhanden war.

„Geniestreich" von Zilog

Die kleinen Rechner fanden aber zunehmend Verbreitung, und CP/M unterstütze den Umgang mit Dateien sehr wirksam. Das sicherte dem 8080 lange Zeit die Position als Marktführer und verhinderte, daß sich der 6800 durchsetzen konnte. Trotz zahlreicher Ansätze, ein vergleichbares DOS für den 6800 zu entwikkeln, war der Anschluß an die Umsatzzahlen des 8080 nicht zu schaffen.

Im Verlauf des weiteren Elektronik-Booms versuchten zwar etliche Hersteller, neue Prozessortypen einzuführen; sie scheiterten aber meist am Widerstand des Marktes, auf dem sich nur behaupten konnte, was eindeutige Vorteile aufwies. Auch die in Hard- und Software-Entwicklung getätigten Investitionen verhinderten das Aufkommen neuer inkompatibler Prozessoren.

Nur dem Z80 verhalf eine Art Geniestreich zum Druchbruch. Bei der Firma Zilog, einer Neugründung früherer 8080-Entwickler von Intel, nutzte man die Chance, den Befehlssatz des 8080 zu erweitern. Der 8080 machte nämlich nicht von allen verfügbaren Bitmustern Gebrauch. Unter Verwendung der „freien" Kommandos entstand bei Zilog ein Prozessor mit erheblich höherer Leistung, der aber noch alle 8080-Befehle verstehen und daher mit der vorhandenen Software arbeiten konnte.

Neben dem erweiterten Befehlssatz bot der Z80 ein weiteres wichtiges Plus: Während der 8080 zusätzlich einen externen Taktgeber und einen System-Steuerchip benötigte, hatten die Zilog-Entwickler die gesamte Prozessor-Logik

Ahnentafel

Die gängigen Microprozessoren lassen sich zwei Entwicklungslinien zuordnen, die von den Intel-Chips 4004 und 8008 bzw. vom Motorola 6800 ausgehen.
Diese Tafel veranschaulicht die Entwicklungsstufen und zeigt einige der zugehörigen Rechner, wobei die unbekannteren Prozessoren vielfach auch in wenig verbreiteten Rechnern stekken. Der Apple III ist wohl der einzige Bürorechner mit einem 6502 und der Olivetti M20 die einzige Universalmaschine mit einem Z8000. Beide Computer konnten sich aufgrund des geringen Softwareangebots nicht durchsetzen.

Intel

4004

8008

4004 und 8008: Ursprünglich als kompakter Ersatz für ganze Platinen voller TTL-Gatter gedacht, war der 4004 nur für 4-Bit-Parallelverarbeitung ausgelegt. Intel schaffte mit dem 8008 aber bald den Schritt in den 8-Bit-Bereich. Dieser Chip wurde von Bastlern und Profis schnell als ideale Basis für den Selbstbau von Microcomputern erkannt.

Motorola

6800

6800: Der große Konkurrent des 8080 mit einer völlig anderen Struktur bei vergleichbarer Leistung. Daher prägten sich auch bei den Anwendern zwei Richtungen aus: Die einen gaben dem Konzept des Intel 8080 den Vorzug, die andern schlossen sich der 6800er-Philosophie von Motorola an.

6809

6809: Motorolas eigene Weiterentwicklung des 6800 gilt als der leistungsfähigste 8-Bit-Prozessor. Leider kam er zu spät und wurde nur bei wenigen Rechnern eingesetzt.

Dragon 32

68000

68000: Dem 16-Bit-Prozessor von Motorola blieb der große Erfolg versagt, weil der 8088 das Feld bereits beherrschte. Immerhin hat sich Sinclair beim QL für die bescheidenere Version 68008 entschieden.

68008

Sinclair QL

Apple Lisa

MOS Technology

6502

Commodore PET

6502: Der 8-Bit-Chip von MOS-Technology ist mit dem 6800 zwar eng verwandt, aber nicht kompatibel. Der niedrige Preis machte den 6502 für Hobbyisten und Entwickler interessant. Er ist unter anderem bei den erfolgreichen PET- und Apple-Computern anzutreffen. Auch heute wird er bei Heimcomputern viel eingesetzt, etwa von Oric und Atari.

Apple III

IBM PC

8080

8085

8080: Der erste Prozessor, der nach Erscheinen des CP/M-Betriebssystems die Entwicklung von Heimcomputern und Bürorechnern möglich machte. Daraus entstand bei Intel selbst der 8085, der mehr leistet und weniger externe Chips benötigt. Davon gibt es eine stromsparende CMOS-Version für tragbare Geräte.

Tandy TRS80 Modell 100

8086

8088

8086 und 8088: Der 8088 ist eine abgemagerte Version des 8086, die mit älteren Zusatzbausteinen funktionsfähig ist und deshalb eine Zeitlang sehr beliebt war. Der Einsatz beim Sirius und beim IBM PC machte den 8088 zum verbreitetsten 16-Bit-Prozessor. Bei den meisten neuen Rechnern wird inzwischen der leistungsfähigere Chip 8086 verwendet.

Zilog

Z80

Z80: Eine verbesserte 8080-Version von Intel-Entwicklern, die sich unter dem Namen Zilog selbstständig machten.

Tandy TRS80 Modell 1

Z8000

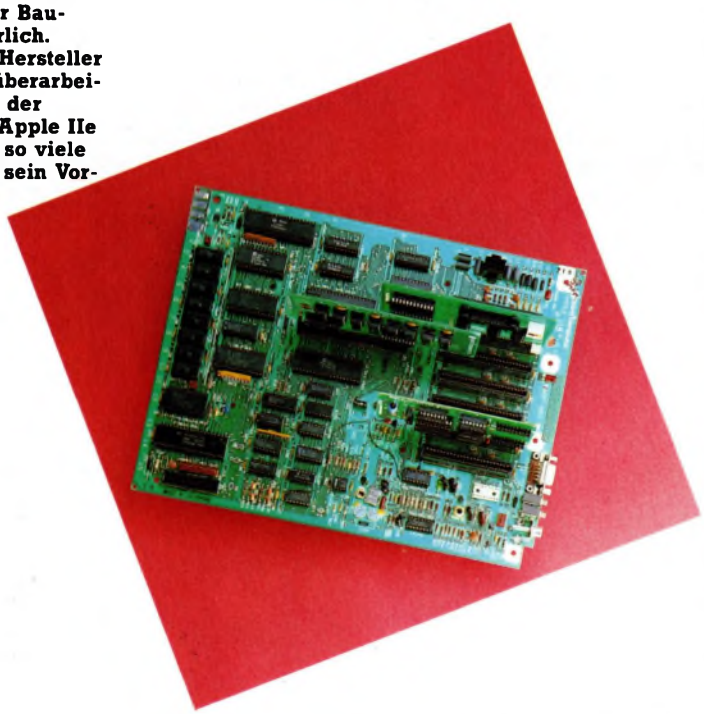
Z8000: Zilogs erster 16-Bit-Prozessor hat sich auf dem Markt nicht durchgesetzt, auch durch den Einsatz des 8088 bei IBM bedingt. Zilogs zweiter 16-Bit-Anlauf ist der Z800, der volle Kompatibilität mit dem Z80 (also auch dem 8080) verspricht.

Olivetti M20

in einen einzigen Chip gesteckt. Das machte den Z80 für Rechner-Hersteller trotz des relativ hohen Preises sehr attraktiv.

Obwohl sein Marktanteil sich in Grenzen hielt, hatte der 6800 bei einer Reihe von Entwicklern und Programmierern durchaus seinen festen Platz. Das veranlaßte Motorola schließlich zum Bau eines „hochgezüchteten" 8-Bit-Nachfolgers mit der Bezeichnung 6809. Unglücklicherweise kam fast gleichzeitig ein Konkurrent namens MOS-Technology ebenfalls mit einer verbesserten 6800-Version heraus, nämlich dem 6502. Dieser Prozessor ist der bekannteste aus der 6500er-Reihe, deren Vertreter trotz unterschiedlicher Ausstattung und Leistung alle mit dem gleichen Befehlssatz arbeiten.

Der 6502 ist zwar vom Konzept her eng mit dem 6800 verwandt, aber nicht mit ihm kompatibel – weder in bezug auf die Hardwareanforderungen noch auf die Software. Dagegen beinhaltet der Z80 den gesamten Befehlssatz des 8080 und kann ihn – vorausgesetzt, die CPU-Platine wurde entsprechend modifiziert – auch softwaremäßig ersetzen.



Bei der Verwendung hochentwickelter Chips sind zur Erfüllung der gleichen Funktionen immer weniger Bausteine erforderlich. Nachdem der Hersteller den Apple II überarbeitet hatte, wies der Nachfolgetyp Apple IIe nur noch halb so viele Chips auf wie sein Vorgänger.

Mit dem Befehlssatz des 6502 ist jeder 6800-Programmierer sofort vertraut. Der 6502 bietet gegenüber dem 6800 mehr Möglichkeiten und bereitet weniger Schnittstellenprobleme, ist aber eben nicht kompatibel. Deshalb läßt sich die gegenwärtige Marktposition des 6502 nur so erklären: Der 6502 hatte das Glück, als „Mitreisender" vom Senkrechtstart der Apple-Rechner zu profitieren.

Bei Erscheinen des ersten Apple war für Schreibtisch-Rechner noch das Steckkarten-Konzept vorherrschend: Die Systemkomponenten wurden über S-100-Stecker (100polig) auf ein „Motherboard" mit Bus- und Versorgungsleitungen aufgesteckt. Das Grundsystem bestand aus Netzteil, Motherboard, CPU-Karte, Speicherkarte, Videokarte, dazu meist noch Drucker-Interface-Karte und Floppy-Steuerkarte – kein Wunder, daß solch ein Rechner viel teurer sein mußte als ein Ein-Platinen-Computer wie der Apple.

Trotz des günstigen Preises kam der eigentliche Durchbruch für die Apple-Mannschaft unter Stephan Wozniak erst mit dem VisiCalc-Softwarepaket. Dieses Programm fand bei Geschäftsleuten großen Anklang, weil sich finanzielle Planungen damit viel schneller und müheloser als mit Taschenrechner, Bleistift und Papier durchführen ließen. VisiCalc war so erfolgreich, daß es Apple enorme Verkaufszahlen einbrachte und gleichzeitig den 6502 als einen der führenden Microprozessoren etablierte. Auch Commodore wählte für den PET und dessen Nachfolger den 6502.

Während sich bei den 8-Bit-Rechnern der 6502 behaupten konnte, hob sich schon der Vorhang für die 16-Bit-Computer. Intel offerierte dafür den 8086 und 8088, Motorola den 68000 und Zilog den Z8000. Alle drei Prozessoren haben ihre Stärken, aber keiner ist mit dem jeweiligen 8-Bit-Vorläufer kompatibel. Zum Glück für Intel kamen Digital Research und Microsoft sehr schnell mit Betriebssystemen für den 8086 und 8088 heraus (CP/M–86 bzw. MS-DOS), während Zilog und Motorola von den Sotwarehäusern arg vernachlässigt wurden. Vorteilhaft für die Intel-Prozessoren wirkte sich auch aus, daß IBM seinen Personal Computer mit dem 8088 bestückte.

Neuer Kampf um den Markt

Der Kampf um den 16-Bit-Prozessormarkt scheint eine Neuauflage der 8-Bit-Geschichte zu werden. Intels 8086 und der 8088 sind ähnlich wie zuvor der Z80 und der 6502 zum Standard geworden, wofür die Software-Unterstützung durch CP/M-86 und MS-DOS sowie der Einsatz in Bestsellern wie dem IBM PC und dem Sirius entscheidend waren. Der Z8000 von Zilog wurde nur bei einem Universalrechner, nämlich dem Olivetti M20 verwendet. Olivetti gelang es aber nicht, die entsprechende Software zu erstellen, und mußte schließlich das Gerät mit einer 8086-Steckkarte aufrüsten, um darauf MS-DOS und CP/M-86 fahren zu können. Daraufhin nahm Zilog eine 16-Bit-Neuentwicklung namens Z800 in Angriff, die mit dem Z80 softwarekompatibel ist.

Trotz der jetzigen Expansion im 16-Bit-Bereich arbeitet noch die Mehrheit der zur Zeit gehandelten Rechner mit Z80- oder 6502-Prozessoren. Gegenüber ihren Vorgängern bieten die 16-Bit-Maschinen zweifellos mehr Leistung und Geschwindigkeit, aber auch in den 8-Bit-Rechnern steckt noch sehr viel Leben, wenn man das vorhandene, sehr große Softwareangebot berücksichtigt.

Zu neuen Ufern

Das Unternehmen Psion produziert vornehmlich für Sinclair Research. Es hat das Programm „Horizons" für den Spectrum entwickelt und die vier Applikationen, die im Lieferumfang des QL enthalten sind. Unlängst wandte man sich dem Business-Software-Bereich zu und stieg in den Hardware-Bereich ein.

Psions Xchange ist ein Paket integrierter Business-Software, das auf Programmen basiert, die für den Sinclair QL entwickelt wurden. Zu Xchange gehören die Textverarbeitung Quill, die Datenverwaltung Archive, das Finanzplanungsprogramm Abacus und das Grafiksystem Easel. Die vier Programme können sowohl als Paket als auch einzeln erworben werden. Xchange gibt es für den IBM PC und PC XT, ACT Apricot und Apricot XI sowie den Sirius I. Zusätzliche Versionen für den Apple Macintosh und den DEC Rainbow sind in Vorbereitung.

Psion wurde 1981 von Dr. David Potter, einem Dozenten des Imperial College, London, gegründet. Erster Marketing-Coup des Jungunternehmens war die Veröffentlichung von vier Programmen für den ZX 81: Flug-Simulator, Backgammon, Vu-Calc und Vu-File. Charles Davies und Colly Myers zeichneten als Programmautoren verantwortlich. Mit diesem Angebotsspektrum hochwertiger Software wurde der Name des Hauses schlagartig bekannt. Als Sinclair Research 1982 einen Hersteller zur Entwicklung eines Demo-Programmpaketes suchte, das die Stärken des Spectrum aufzeigen sollte, war es klar, daß Psion das betreffende Unternehmen war.

Die Popularität der Sinclair-Rechner bedeutet: Es gibt ein großes Absatzgebiet für Psion-Software. Die Firma hatte eine Reihe bemerkenswerter Erfolge. So wurden über 500 000 Exemplare des „Flugsimulator" für den ZX 81 und den Spectrum verkauft.

Innovativ war Psion schon immer. Das Haus selbst trug entscheidend zur Entwicklung des sogenannten „Kreuz-kompilierens" als Software-Schreibtechnik für den Heimcomputer bei. Das Einführungspaket „Horizons" für den Spectrum wurde auf einem Tandy TRS-80 entwickelt. Heute erfolgt die Herstellung sämtlicher Software auf zwei VAX 750-Minicomputern. Psion hat zudem QLUB organisiert, eine Hotline für QL-Anwender, die bei schriftlichen Reklamationen innerhalb 48 Stunden Antwort und Lösungen garantiert.

Dr. David Potter ist Gründer und Hauptanteilseigner von Psion. Der Ex-Akademiker, der am Imperial College in London auf Computerphysik spezialisiert war, gründete Psion als Potter Scientific Investments.

Der Psion Organiser

Die Pläne für eine Reihe von Geschäftsprogrammen für den IBM PC, Apricot, Sirius, DEC Rainbow und Macintosh-Computer sind weit fortgeschritten. Wie auch beim QL-Paket gehören zur Xchange-Programmpalette Spreadsheet, Datenbank, Grafikprogramm und Textverarbeitung. Der Unterschied zwischen herkömmlichen Programmen besteht nach Auskunft von Psion darin, daß die Daten untereinander ausgetauscht werden können.

Eine weitere, große Aufmerksamkeit erregende Entwicklung von Psion ist der Organiser. Viele Leute waren überrascht, daß ein Haus, das eigentlich als Software-Unternehmen gesehen wurde, einen Taschencomputer entwickelte. Robin Kinnear, ein Sprecher der Firma, versucht, die richtige Perspektive in diese Entwicklung zu bringen: „Hintergrund ist, daß Psion ein Microcomputer-Software-Unternehmen ist. Unter dem Gesichtspunkt Software-Paket hielten wir den ‚Organiser' für eine sehr gute Idee. Doch wir stellten fest, daß es derartiges nicht gab. Also beschloß Psion, eigene Hardware zu entwickeln. Diese Entwicklung war stark durch Software geprägt."

Derzeit sind nur drei Applikationen (Wissenschaft, Mathematik und Finanzen) für den Organiser erhältlich, neben den acht-KByte- und 16-KByte-RAM-Paketen (oder „datapaks", wie sie genannt werden). Psion wird weitere Programme entwickeln und wurde bereits von vielen Interessenten angesprochen, die Software für diesen Rechner schreiben wollen.

Gegenwärtig ist Expansion im Markt eine der Prioritäten des Unternehmens: Jüngst hat man Niederlassungen in den USA und Südafrika etabliert, Verträge zur Verbesserung des Psion-Produktvertriebs in Europa wurden unterzeichnet. Weiterhin hat Sinclair Research mit dem Verkauf von Rechnern nach Osteuropa begonnen, angefangen mit einer Lieferung von 400 ZX 81 in die Tschechoslowakei. Psion, das sich mit der Entwicklung fremdsprachiger Versionen der eigenen Software befaßt, wird diesem Vorbild sicherlich bald folgen und hier ebenfalls einsteigen.

Qualitätskontrolle

Die Firma Softsel ist weltweit einer der einflußreichsten Großhändler von Computer-Hard- und Software. Wenn Sie kommerzielle Programme oder Computerspiele amerikanischer Hersteller einsetzen, ist die Software mit großer Wahrscheinlichkeit auch durch die Hände von Softsel gegangen.

Bei der ständig wachsenden Zahl neuer Programmpakete ist es für den Software-Händler unmöglich, alle neuen Produkte persönlich zu beurteilen, da das ausführliche Testen sehr zeitaufwendig ist. Eine zu flüchtige Bewertung birgt jedoch immer die Gefahr, daß man unverkäufliche und zu teure Pakete einkauft. Die Firma Softsel nimmt Händlern dieses Risiko ab, indem sie jedes angebotene Programmpaket ausführlich testet, bevor sie es in ihren Katalog aufnimmt.

Simon Rhodes, Softsels Marketingdirektor für Großbritannien, erläutert den Ablauf folgendermaßen: „Neue Programmpakete werden zunächst von der technischen Abteilung auf ihre Benutzerfreundlichkeit hin untersucht – es wird getestet, ob das Paket gut programmiert ist, ob die Dokumentation in Ordnung ist und die Grafik anspricht. Danach beurteilt unsere Werbe- und Verkaufsabteilung, ob das Programm mit genügend Werbeaufwand gefördert wird."

Außer Qualitätskontrollen bietet Softsel noch Verkaufsförderung und die Rücknahme nicht verkaufter Programme. Zwar sind die Programme direkt bei den Herstellern billiger, durch Großeinkäufe kann Softsel jedoch Rabatte bieten, die diesen Preisunterschied auf ein Minimum reduzieren. Simon Rhodes sagt dazu: „Natürlich erhalten die Händler einen besseren Preis, wenn sie direkt bei den Herstellern kaufen. Auf lange Sicht gesehen ist es jedoch teurer, da sie mit Hunderten verschiedener Firmen verhandeln müssen, statt nur mit einem Unternehmen."

Herb Blumstein, Geschäftsführer

Simon Rhodes, Marketingdirektor

Gründung in Amerika

Softsel wurde 1980 von Robert Leff und David Wagman gegründet, die früher in der Datenverarbeitung von Transaction Technology – einer Tochtergesellschaft des Finanzierungsgiganten Citicorp – zusammengearbeitet hatten. Das rapide Wachstum von Softsel zeigte, daß Leff und Wagman mit ihrem Konzept genau richtig lagen. Vier Jahre nach der Gründung beschäftigte Softsel weltweit 350 Mitarbeiter und setzte im letzten Geschäftsjahr 87 Millionen Dollar um. Allein in den Vereinigten Staaten unterhält die Firma vier große Läger – Atlanta, Chicago, Los Angeles und New York – und bietet den Händlern landesweit 4500 Programmpakete an.

Im September 1982 streckte Softsel ihre Fühler auch nach England aus und gründete schon ein halbes Jahr später das Zweigunternehmen mit dem Namen „Softsel Computer Products". SCP hat seinen Sitz in Feltham, in der Nähe des Flughafens Heathrow, und beliefert Europa und den Mittleren Osten mit mehr als 2500 verschiedenen Produkten.

Umfangreiches Programmangebot

Die Zukunft der Firma scheint gesichert. So plant die englische Tochtergesellschaft eine Ausweitung der kommerziellen Software, die augenblicklich etwa die Hälfte ihres Katalogs ausmacht. Softsel wird aber auch den zweiten Hauptbereich der Softwareproduktion – die Spielprogramme – nicht vernachlässigen.

Auch auf die übrigen europäischen Länder möchte Softsel seinen Einfluß ausdehnen. So gibt es in Deutschland eine Filiale mit Sitz in München, und auch die Computerfreaks in Frankreich und Italien profitieren von dem umfassenden Programmangebot.

Bug-Byte

Der Aufstieg der Firma Bug-Byte ist eng an die Entwicklung des englischen Heimcomputermarkts gebunden. Aus dem Hersteller von Sinclair-Programmen wurde einer der führenden Spielelieferanten für die populärsten Maschinen. Zu seinen Bestsellern gehören Manic Miner und Twin Kingdom Valley.

Bug-Byte wurde im Frühjahr 1980 gegründet, als die beiden Chemiestudenten Tony Baden und Tony Milner ihre ersten Programme für den neuerstandenen ZX80 schrieben. Bald merkten sie, daß es für dieses Gerät kaum Software gab, und entschieden sich, ihre eigenen Programme anzubieten. Sie kauften 40 Leercassetten und annoncierten per Computermagazin ein Softwarepaket mit fünf Spielen. Anfangs gingen pro Woche fünfzehn Bestellungen ein. Die beiden Partner investierten den Gewinn in weitere Anzeigen und die Erstellung neuer ZX80-Programme.

Mit dem Erscheinen des ZX81 Anfang 1981 ging die Nachfrage für ZX80-Programme rapide zurück. Baden und Milner wandten sich nun der Erstellung von ZX81-Software zu. Nach ihrem Studienabschluß im Juni 1981 verlegten sie Bug-Byte in Tony Badens Heimatstadt Liverpool. Sie betrieben die Firma nun auf Vollzeitbasis, und die Anzahl der verkauften Programme verdoppelte sich in kurzer Zeit.

Weihnachten 1981 wurde der Konkurrenzkampf auf dem Spielemarkt härter. Zur Sicherung ihrer Verkaufszahlen beauftragte Bug-Byte eine Werbeagentur und gestaltete die Cassetteneinlagen und Anzeigen von nun an farbig. Eine professionelle Kopierfirma sorgte für die hohe Cassettenqualität.

Mit der neuen Aufmachung ihrer Produkte, die inzwischen über ein landesweites Händlernetz vertrieben wurden, stiegen die Verkaufszahlen sprunghaft an, und die Firma stellte weitere Mitarbeiter ein. Auch durch die zahlreichen neuen Computermodelle wuchs die Nachfrage. Bug-Byte beschäftigte daher freie Mitarbeiter, um den Bedarf decken zu können. Viele dieser Mitarbeiter gründeten später Konkurrenzunternehmen wie Quicksilva und Software Projects. Pro Cassette zahlt Bug-Byte seinen Programmierern einen festen Anteil. Spiele, die in der Bestseller-Hitparade unter den ersten zwanzig sind, bringen dem Autor nach Aussage der Firma im ersten Jahr zwischen 40 000 und 150 000 Mark ein.

Eigene Kopierfirma

Ende 1982 verkaufte Bug-Byte seine Programme außer über die großen Ladenketten auch über 200 unabhängige Läden und stellte nach und nach seinen Versandhandel ein. Da die Kopierfirmen nicht genügend Cassetten für das Weihnachtsgeschäft herstellen konnten, gründete Bug-Byte eine eigene Kopierfirma mit dem Namen Spool. 1983 bezog Bug-Byte dann neue Räume in Liverpool.

Zu den Bestsellern von Bug-Byte zählen das Abenteuerspiel Twin Kingdom Valley (für den Acorn B und Commodore 64) und Manic Miner für den ZX Spectrum und den Commodore 64. Matthew Smith, der Autor von Manic Miner, hat inzwischen die Firma verlassen, um Software Projects zu gründen, und nahm das Copyright seines Spiels mit.

Das Wachstum von Bug-Byte setzt sich auch weiterhin fort. Inzwischen wird seine Software in fast allen westeuropäischen Ländern, Australien, Neuseeland und Südafrika verkauft. Ein kürzlich mit CBS UK geschlossener Vertrag könnte eine Ausweitung in den lukrativen amerikanischen Markt bedeuten.

Tony Milner

Tony Baden

Bug-Bytes Zentrale in Liverpool

Die Weltfirma

Die japanische Firmengruppe Sharp zeichnet sich durch eine umfangreiche Produktpalette aus, die neben Transistoren, Computern und Industrierobotern auch Mikrowellenherde umfaßt. Im Jahre 1983 stieg die Firma auch in den europäischen Heimcomputermarkt ein.

Die Sharp Corporation beteiligte sich seit jeher an technologischen Zukunftsprojekten. Dabei wirkte ihr erstes erfolgreiches Produkt eher alltäglich – der „Ever Sharp"-Drehbleistift. Sein Erfinder Tokuji Hayakawa gründete die Firma 1915 ausschließlich für die Herstellung seines Schreibgeräts. Dabei blieb es allerdings nicht lange. 1925 brachte Sharp ein Detektorradio auf den Markt. Der Konsumgütermarkt – speziell Fernsehgeräte – wurde für das Unternehmen nach dem Zweiten Weltkrieg immer wichtiger. Mitte der 60er Jahre erfolgte mit einer Serie von Tischrechnern der Einstieg in den kommerziellen Bereich. Heute ist Sharp ein multinationaler Konzern mit sechs Unternehmensbereichen, 34 Produktionsstätten in 30 Ländern und circa 45 000 Mitarbeitern.

Der erste Sharp-Computer MZ80K wurde ab 1981 in Europa angeboten. Bereits ein Jahr später folgten die Rechner MZ80A und MZ80B. Die Geräte waren zwar für den kommerziellen Einsatz entwickelt, fanden aber auch unter den Heimcomputer-Freunden treue Anhänger. Alle Modelle hatten einen eingebauten Monitor und ein Cassettenlaufwerk. Die Sharp-Rechner waren „Clean Machines" – so genannt, weil sie keine Sprache im ROM hatten. Das war vorteilhaft, da sich so unterschiedliche Sprachen und Betriebssysteme (einschließlich CP/M) problemlos von Cassette laden lassen.

Die Ergebnisse der Forschung aus dem firmeneigenen Technologiezentrum werden an die Abteilung „Industriegeräte" im japanischen Yamato-Koriyama-shi bei Nara weitergegeben (Bild unten). Hier werden alle Sharp-Produkte – einschließlich Taschenrechner und Heimcomputer – zur Produktionsreife gebracht.

In den folgenden Jahren hat Sharp sein Angebot stark erweitert: Zuerst mit dem Profi-Rechner MZ-3541, später durch den Pocket-Computer PC-1500. Der Erfolg des PC-1500 führte zur Einführung des noch kompakteren PC-2500 – ebenfalls erfolgreich.

Bei den Heimcomputern ist Sharp mit den MZ700- und MZ800-Serien vertreten. Die in Europa verkauften Modelle stechen besonders durch ihre leistungsstarke Grafik heraus. Der dafür verwendete Speicherplatz enthält in der „Nippon-Version" lediglich den umfangreichen japanischen Zeichensatz. Die Geräte verfügen je nach Typ über eingebaute Cassettenlaufwerke und Vierfarb-Plotter.

Weiteres Wachstum

Im Mai 1984 erschien mit dem PC-1500A ein verbessertes Modell des PC-1500. Das Gerät ist mit 8,9 KByte RAM ausgestattet und kann bis 24 KByte aufgerüstet werden. Nicht lange danach erschien der PC-1350, ein Pocketcomputer mit einer vierzeiligen LCD-Anzeige und Grafikmöglichkeiten.

Auf die Frage nach weiteren Plänen sagt Rod Goodier, Verkaufsleiter in England: „Interessante Möglichkeiten sehen wir bei den Pocket-Computern. In diesem Bereich möchten wir sehr gern expandieren." Aber auch der Heimcomputer-Markt soll nicht zurückstehen: „Mit der MZ700-Serie haben wir einen guten Start gehabt. Wir werden auf jeden Fall in diesem Markt bleiben!" Der englische Unternehmenssprecher Peter Fletcher erklärt: „Unsere Abteilung für Büro- und Heimcomputer ist relativ jung. In England erzielt sie bisher nur 25% vom Umsatz, wir hoffen aber auf ein Wachstum des Anteils am Gesamtergebnis der Firma."

Sharp erweitert mittlerweile seine Produktionskapazität in Europa. In Wrexham/England entstand für mehr als 40 Millionen DM eine neue Fabrik, in der Videorecorder hergestellt werden. Das ehrgeizig gesteckte Ziel liegt für 1985 bei immerhin 60 000 Geräten, die in ganz Europa verkauft werden.

Auf die Frage, ob Sharp sich am MSX-Standard der anderen japanischen Hersteller beteiligen wird, antwortet Rod Goodier: „Natürlich haben auch wir ein MSX-Gerät entwickelt. Es steht aber noch nicht fest, ob und wann es auf den Markt kommt."

Absolut super

Die Cray-Familie der Supercomputer bildet die Elite der Großrechner. Industrie, Universitäten, Forschungslaboratorien der Regierungen sind Abnehmer dieser Rechner – ob nun Explosionen simuliert, Maschinen getestet oder Naturereignisse vorhergesagt werden sollen.

Das Erscheinungsbild des Cray-1 ist unverwechselbar. Der Computer befindet sich in einem kreisförmig angeordneten Gebilde. Die Stromversorgung liegt unter den runden Sitzbänken.

Crays Fertigungsstätte in Chippewa Falls, Wisconsin. Hier werden hauptsächlich Chips hergestellt. Doch inzwischen beschäftigt sich Cray stark mit der Gallium-Arsenid-Technik.

Ein Computer-Hersteller, der nach dem Verkauf von nur 16 Systemen zum Jahresende eine Rekordbilanz ziehen kann, muß ein recht ungewöhnliches Produkt herstellen. – So geschehen 1983, als Cray Research den Jahresbericht veröffentlichte und damit die Anzahl der weltweit installierten milliardenteuren Cray-Computer auf 65 gestiegen war.

Das Unternehmen wurde 1972 von Seymour R. Cray gegründet. Er sah damals eine wachsende Nachfrage nach leistungsfähigen Großrechnern voraus. Aufgrund dieser Erkenntnis setzte Cray Research 1983 rund 170 Millionen Dollar um, was einer Verzehnfachung des Umsatzes von vor fünf Jahren entsprach.

Dieser Erfolg basierte auf dem ersten Produkt des Unternehmens, dem Cray-1. Erstmals 1976 installiert, war dieser Rechner für Jahre der leistungsfähigste Computer der Welt. Rein äußerlich wirkte er sehr unauffällig.

Der billigste Cray-1/M-Großrechner kostet heute etwa 16 Millionen Mark. Einer der Hauptgründe für eine so kostspielige Investition ist seine Simulationsfähigkeit. Zeit und Kosten für das Konstruieren von einem Auto-, Schiff- oder Flugzeug-Modell nacheinander und die sich daran anschließenden Tests können gewaltige Dimensionen haben. Der Computer wird dazu eingesetzt, ein „Modell" zu konstruieren, das nur im Speicher des Rechners besteht, modifiziert wird und entsprechend bestimmter Variablenwerte verändert werden kann. Das so erzeugte Modell läßt sich unter allen erdenklichen Voraussetzungen und Bedingungen testen, selbst in Extremsituationen, die physikalisch nicht herstellbar sind. Zu den Cray-Kunden gehören Firmen wie Lockheed und General Motors. Die Systeme machten sich dort bereits nach wenigen Wochen bezahlt. Cray selbst stellt dazu fest, daß sich beispielsweise die Entwicklungskosten und -zeiten für ein neues Flugzeug mit einem seiner Rechner auf die Hälfte reduzieren lassen.

Simulationen sind auch für Wissenschaftler, die Vorhersagen von Naturereignissen treffen wollen, ungeheuer wichtig. So benutzt zum Beispiel auch das britische Wetteramt für seine Vorhersagen einen Cray-1. Aus Daten, die Lufttemperatur, Luftdruck, Feuchtigkeit etc. an verschiedenen Orten und in verschiedenen Höhen darstellen, berechnet der Computer die Bewegung der Luftmassen und die Veränderungen dieser physikalischen Größen über Stunden im voraus. Das beinhaltet eine Vielzahl von Berechnungen, deren Einzelergebnisse die Folgeergebnisse beeinflussen.

Ähnlich umfassend ist die Simulation bestimmter Geschehnisse für Atomforscher, Seismologen und Wissenschaftler in vielen anderen Bereichen. Cray Research ist der Überzeugung, daß die zunehmende Verwendung von Computersimulationen die herkömmliche physikalische Simulation (den Laborversuch) ablöst und so eine neue Ära der Forschung und des Ingenieurwesens einleitet.

Cray beschleunigt diese Revolution durch noch beeindruckendere Produkte. Die Cray-1/S-Serie wurde von der leistungsfähigeren Cray-1/M-Linie abgelöst. Die X-MP-Serie ist noch leistungsfähiger. Diese Rechner verfügen über ein, zwei, drei oder noch mehr Zentraleinheiten und kosten bis zu 50 Millionen Mark. Die Speicherkapazität umfaßt bis zu acht Millionen 64-Bit-Wörter. Der verwendete Massenspeicher wirkt sich sogar auf Entwicklungen im Heimcomputerbereich aus. Crays Diskettenstationen können bis zu 10 Megabyte Daten pro Sekunde übertragen.

Und wem das immer noch nicht schnell genug ist, der bedient sich Crays SSD (Solid-State Storage Device) mit einer Übertragungsgeschwindigkeit von bis zu 2000 Megabyte pro Sekunde aus dem Gigabyte-Speicher. Mit derartigen Maschinen wird Cray Research seine Spitzenposition als Computer-Wegweiser sicherlich weiter ausbauen.

Spezialanwendungen

Für Behinderte bedeutet der Microcomputer mehr als eine Lernverbesserung: Er hilft, die Barrieren zur „normalen" Welt zu überwinden.

Viele Erwachsene und Kinder brauchen spezielle Ausbildungstechniken. Ursache dafür mag eine körperliche Behinderung wie Taubheit oder Blindheit sein oder eine geistige Behinderung. Einige daraus resultierende Bedürfnisse sind offensichtlich, andere hingegen schwer zu erkennen und zu verstehen. Jede Krankheit ist mit speziellen Problemen verbunden, die häufig mehr durch die Verhaltensweise der Gesellschaft als durch die Behinderung selbst geschaffen werden.

Leider wenden die meisten Regierungen weniger Mittel für behindertengerechte Sonderschulen auf als für normale Schulen. Folglich ist dieser Markt für Hard- und Software-Entwickler nicht lukrativ. Die Energie und Hingabe derjenigen, die dennoch in und für diesen Bereich arbeiten, sollten als beispielhaft für den gesamtpädagogischen Bereich dienen. So sammelten Schüler in South Yorkshire umgerechnet 40 000 Mark in ihrer Freizeit, um die Entwicklung eines Spezialtelefons für Hörbehinderte zu ermöglichen. Diese Arbeit wird üblicherweise „an der Basis" geleistet, von Einzelpersonen und Initiativ-Gruppen. Lehrer, Programmierer und Ingenieure arbeiten in Eigeninitiative und entwickeln Hard- und Software für bestimmte Ausbildungszwecke.

Lehrerausbildung

In Manchester wurde ein Special Education Microelectronics Resources Centre (SEMERCs) gegründet, ebenso in Newcastle und Redbridge, um den speziellen Bedürfnissen Behinderter in der Ausbildung zu entsprechen. Eine der Hauptaufgaben besteht darin, Lehrer an Spezialschulen auszubilden und sie über wichtige technische Entwicklungen und Neuerungen zu informieren.

Die Situation wird durch die vielgestaltigen Probleme der speziellen Erziehung noch komplizierter. Bei einigen Krankheiten ist die Kommunikation das größte Problem. So kann ein intelligenter, aufgeweckter Mensch durch eine körperliche Behinderung unfähig sein, mit der Außenwelt zu kommunizieren. Die Microtechnologie hat vielen Menschen neue Kommunikationskanäle eröffnet. So können etwa mit Hilfe der „Photonic Wand" (einem optischen Sensor, der sich auf einem Plastikhelm befindet und durch Kopfbewegung gesteuert wird) stumme Menschen, die ihre Gliedmaßen nicht bewegen können, einen Acorn B bedienen. Der optische Sensor ist einem Lichtgriffel ähnlich und bewegt einen Cursor über den Bildschirm – entsprechend den Kopfbewegungen.

Am „Vincent"-Arbeitsplatz können Menschen mit verschiedenen Behinderungen arbeiten. Mit dem „Perkins Brailler" können Blinde Informationen in den Computer eingeben. Der Sprachsynthesizer ermöglicht eine hörbare Rückkopplung. Die Textdarstellung auf dem Monitor wurde für Sehbehinderte vergrößert. Der Arbeitsplatz ist außerdem mit einer „Konzept-Tastatur" verbunden. An die im Bild gezeigte Station kann auch ein spezieller Blindenschrift-Drucker angeschlossen werden.

Dieses Hilfsmittel kann auch von Menschen benutzt werden, die ihre Handbewegungen nicht steuern können. Im SEMERC in Manchester hat man das Photonic Wand mit einem Sprachsynthesizer verbunden. Für Telefongespräche entwickelte Antworten wie „Können Sie den Satz wiederholen" werden auf dem Bildschirm dargestellt und von einem Sprachsynthesizer ausgesprochen. Man hofft, den Wortschatz erweitern zu können und eine einfache Methode zu entwickeln, um den Synthesizer mit dem Telefon zu verbinden.

Computer sind für Sehende entwickelt worden. Monitore, Drucker und Plotter sind visuelle Ausgabeeinheiten, und alle Programme bedienen sich der visuellen Darstellung. Bei

den Projekten „Open University's Computing" und „Blind Project" arbeitet man daran, bestehende preiswerte Software auch Blinden zugänglich zu machen. Das Projekt läuft in Zusammenarbeit mit Blinden an Schulen, am Arbeitsplatz und daheim. Die Arbeitsplätze bestehen aus Computer, Diskettenstation, Monitor, Drucker, Sprach-Synthesizer, einer Spezialtastatur und einem abgewandelten „Perkins Brailler" und wurden in Schulen und Arbeitsstätten installiert.

Der Perkins Brailler wurde vor 40 Jahren zur Erzeugung von Blindenschrift erfunden. Dieses Gerät wird mit dem Computer verbunden und ermöglicht so einem Blinden, den Ausdruck zu lesen. Es gibt spezielle Software, mittels der ein Computer Blindenschrift in normale Schrift umwandeln, speichern, editieren und ausdrucken kann.

Für die Verwendung mit der üblichen Tastatur wurden mehrere sprechende Textverarbeitungsprogramme entwickelt. Eingegebene Zeichen und spezielle Tasten werden durch

Die „Photonic"-Ausrüstung erlaubt selbst stark behinderten Menschen, die weder sprechen noch ihre Gliedmaßen bewegen können, einen Acorn B zu steuern. Die Kopf-Bewegung wird in ein Analogsignal umgewandelt, das der Computer verarbeitet.

gesprochene Ausgaben bestätigt. Die „Delete"-Taste „spricht" zugleich die Bezeichnung des gelöschten Zeichens. Text wird mit Hilfe eines Sprech-Cursors editiert. Während er über den Text läuft, werden Wörter oder Sätze gesprochen. Der Cursor kann jederzeit angehalten werden, um Ergänzungen oder Löschungen vorzunehmen.

Manche behinderten Kinder sind nicht in der Lage, eine normale Tastatur zu bedienen. Für solche Menschen wurde ein flaches, berührungsempfindliches Tablett entwickelt. Diese „Konzept-Tastatur" wird vom Lehrer mit austauschbaren Folien versehen. So kann die Tastatur beispielsweise in vier Segmente geteilt werden, die jeweils ein Symbol tragen. Mit ihnen läßt sich eine auf dem Boden befindliche Turtle steuern. Durch Drücken unterschiedlicher Segmente kann die Turtle vorwärts, rückwärts, nach links oder rechts geführt werden. Das ist einfacher als „FD 1 RETURN" usw. einzugeben.

Mit der bei Apples „Macintosh" verwendeten Maus entfällt die „Tastatur-Barriere" ebenfalls. Eine Benutzer-Gruppe namens AMASE (Apple Macintosh Applications in Special Education) beschäftigt sich ausschließlich mit der Erforschung der Möglichkeiten der Maschine in diesem Bereich.

Man hat auch spezielle Schalter entwickelt, die durch verschiedene Körperteile bedient werden können. So gibt es zum Beispiel für spastische Patienten eine Spezialschaltung. Zwei kleine Metallscheiben werden in Augennähe angebracht und können die horizontale Bewegung der Augen registrieren. Die dabei entstehenden elektrischen Signale werden in Steuersignale umgesetzt.

Im Bereich „Science and Society" der Bradford-Universität befestigte man auf einer Turtle ein blinkendes, orangefarbenes Licht, wodurch ein seh- und körperlich behinderter 18jähriger Patient imstande war, sie zu sehen. Durch Verwendung eines Konzept-Keyboards konnte er die Turtle steuern und die Ergebnisse seiner Aktionen sehen. Für diesen jungen Mann, wie für viele behinderte Kinder, stellen solche Erfahrungen oft den ersten Schritt von der Rolle des passiven Beobachters zur bewußten Anteilnahme und Aktion sowie zur Beeinflussung ihrer Umgebung dar.

Auf der British Logo User's Group-Konferenz im Jahre 1984 führte Dr. Sylvia Weir bemerkenswerte Beispiele für die Fortschritte an, die geistig und körperlich behinderte Kinder durch den Umgang mit LOGO machten. Der

17jährige Michael, ein Sonderschüler, hatte noch nie ein Wort geschrieben. Seine Lehrer vermuteten, daß er sehr intelligent sei, aber nicht kommunizieren könne. Nachdem er mit dem Computer vertraut gemacht worden war, programmierte er zehn Stunden täglich. Zwei Jahre später schrieb er an der Universität an einer Arbeit zum Thema „Gefangene Intelligenz". Ein 7½jähriger sprachbehinderter Junge durfte eine Turtle via Knopfbox steuern, einem der „Konzept-Tastatur" ähnlichen Gerät. Er wurde durch dieses Erlebnis so aufgeregt, daß er zum ersten Mal sprach.

Dr. Weir wandte LOGO auch an einer Sonderschule an, wo eine Gruppe von Kindern, die unter zerebraler Paralyse litten, unterrichtet wurde. Die Steuerung des Computers half ihnen, die durch ihre Behinderung entstandene Passivität zu überwinden. Dr. Weir schrieb: „Die daraus resultierenden Verbesserungen im Sinne der Wertigkeit einer Person und die Konsequenzen für intellektuelle Aktivität haben die Schule veranlaßt, ein eigenes Computerzentrum zu installieren."

Als Ergebnis gemeinsamer Arbeit des SEMERC in Manchester und des örtlichen Computerclubs entstand „Micromike", ein Gerät, das sprachgestörten Kindern hilft. Ein modifiziertes Funk-Mikrofon wurde mit einem Acorn B verbunden und erlaubt Kindern, mittels Stimme verschiedenste Aktivitäten auf dem Bildschirm zu steuern. Zu der dafür entwickelten Software gehört „City", ein Programm, mit dem sie die Skyline einer Stadt zeichnen können. Höhe und Breite der Gebäude werden durch Lautstärke und Dauer der Stimme bestimmt. Programme dieser Art bieten den Kindern die Möglichkeit, ihre Stimme auf individuelle Art zu beherrschen.

Wie die allgemeine Pädagogik, so hinkt auch die Spezialpädagogik der technischen Entwicklung weit hinterher. Das resultiert daraus, daß zuwenig Mittel zur Forschung zur Verfügung stehen und daß Hard- wie Software-Hersteller ihren Umsatz mit Geschäfts- und Industrieprogrammen erzielen – nicht mit Ausbildungsprogrammen. Der spezialpädagogische Bereich ist noch kleiner. Die SEMERC-Zentren können die Bitten um Hilfe längst nicht alle erfüllen. Der Mangel an finanziellen Mitteln und Erfahrung verlangsamt die Entwicklung von Hard- und Software an diesen Zentren. Und aus dem Geldmangel an Schulen resultiert eine nur geringe Einbeziehung der Computertechnik zu Ausbildungszwecken.

Die „Konzept-Tastatur" wurde nach dem Prinzip des Grafiktabletts entwickelt. Selbst spastische Anwender können mit dem Computer kommunizieren. Eine Beherrschung der Tastatur ist nicht erforderlich. Die Gestaltung der Folien hilft, die Hemmschwelle gegenüber dem Computer herabzusetzen.

Computer und die Begabten

Begabte und talentierte Kinder haben ebenfalls spezielle pädagogische Bedürfnisse. Der Einsatz von Computern bei der Ausbildung bietet ihnen ebensoviele Möglichkeiten, wie es bei behinderten Kindern der Fall ist.

Behinderte wie begabte Kinder gewinnen mit dem Computer Kontrolle über ihre Umgebung und können so natürliche oder von Menschen geschaffene Barrieren überwinden, um individuell aufzuwachsen und sich selbst zu verwirklichen. Im einzelnen bieten Computer begabten Kindern folgende Möglichkeiten:

- **Entwicklung von Fähigkeiten und Interessen;**
- **kreative neue Wege, Problemlösungen zu finden;**
- **Kommunikation und Zusammenarbeit mit anderen begabten Schülern;**
- **die Möglichkeit, ermüdende und langweilige Übungen auf interessante Art durchzuführen;**
- **ein neues Medium für Experimente.**

Technik macht Druck

Der Einsatz elektronischer Technologie im industriellen Bereich nimmt ständig an Bedeutung zu. Am Beispiel der Druckindustrie zeigen wir die Problematik auf.

Die Behauptung, daß die Computertechnologie unsere Lebens- und Arbeitswelt radikal verändert, ist mittlerweile zu einem Klischee geworden. Besonders deutlich wird diese Entwicklung im Verlagsbereich. Trotz bedeutender technologischer Fortschritte im Bereich der Textverarbeitung und des computerisierten Layouts findet man bei kleinen Tageszeitungen mancherorts noch heute Maschinen, die aus den dreißiger Jahren stammen. Nicht nur die Kosten für neue Technologie begünstigen das Nebeneinander von Alt und Neu. Auch die in diesem Berufszweig traditionell stark gewerkschaftlich organisierte Arbeiterschaft macht sich gegen Rechnereinsatz stark, der reihenweise Arbeitsplätze wegrationalisiert.

Was vermag die Technik überhaupt? Viele Autoren haben ihre Schreibmaschinen durch Mikrocomputer ersetzt. Sie erlauben es ihnen, Texte in einem Arbeitsgang zu schreiben und zu korrigieren, die Arbeitsleistung zu erhöhen und saubere, preiswerte Manuskripte schnell für Verleger zu erstellen.

Ob man einen Rundbrief an Clubmitglieder oder einen Zeitungsartikel schreiben will, ein transportabler Computer ist ein leistungsfähiges Hilfsmittel. Aus vor Ort getippten Notizen wächst ein Manuskript, nach Bedarf versehen mit Fußnoten, automatischer Paginierung, Kapitelüberschriften und Randbemerkungen – alles kann vor dem Ausdruck eingefügt werden, um ein sauberes, einwandfreies Manuskript zu erstellen. Es ist der erste Schritt auf dem langen Weg zum gedruckten Endprodukt.

Bei den meisten heute produzierten Magazinen finden nch immer vorwiegend traditionelle Produktionsprozesse Anwendung. Üblicherweise wird der Autor eines Artikels aufgefordert, ein maschinengeschriebenes Manuskript einzureichen. Es wird dann in der Redaktion überarbeitet, redigiert. Ein Redakteur korrigiert das Manuskript auf inhaltliche Richtigkeit. Das Manuskript wird dann an einen Korrektor weitergeleitet, besser einen weiteren Redakteur, der wiederum handschriftliche Korrekturen anfügt und Teile umschreibt, damit der Stil dem Verlagshaus entspricht. Das Layout nimmt die Auszeichnung

Viele Zeitungen benutzen zur Satzerstellung noch heute Maschinen, die als „Linotype" bekannt sind. Die 1886 erfundene Linotype stellt aus Bleilettern Satzzeilen her. Aus Gründen der Kostenreduzierung findet Computertechnologie vor allem bei Lokalzeitungen Anwendung. Die neuen Maschinen erlauben elektronische Texterstellung.

für die Setzmaschinen vor, um dem Setzer Angaben über Schriftart, Größe, Durchschuß und Zeilenbreite zu geben. Sie erfolgt ebenfalls handschriftlich auf dem Manuskript des Autoren. Aber damit nicht genug.

Fahnenkorrektur

Ist der Artikel gesetzt, ähnelt er fast der druckfertigen Form. Es können allerdings noch Fehler darin stecken, etwa wenn die Setzer bei der Übertragung Fehler gemacht haben. Der abgezogene Artikel wird als erster Fahnenabzug bezeichnet. Der Produktionsredakteur liest ihn wieder sorgfältig und sendet ihn korrigiert an den Satz zurück. Daraufhin wird eine zweite Version erstellt, der zweite, nun fehlerfreie Abzug.

Der Layouter greift wiederum beim Eintreffen der ersten Fahne in den Produktionsprozeß ein. Seine Aufgabe ist es, mit dem Abzug ein Seitenlayout zu erstellen, die Form zu schaffen, in der der Artikel gedruckt erscheint. Natürlich enthalten die meisten Magazine nicht nur Text, sondern auch Fotos und Zeichnungen, zu denen Bildunterschriften und Anmerkungen gehören. Der Layouter muß all diese Komponenten zu einer in sich geschlossenen und zugleich interessant wirkenden Seite komponieren, die dem Auge etwas bietet.

Üblicherweise wird dafür zuerst ein „Scribble", eine Rohskizze, angefertigt, in der die Einzelelemente wie Steine eines Puzzles verschoben werden. Dabei wird zuweilen auch der erste Fahnenabzug ins Layout integriert. Manche Layouter schreiben einfach die Textlänge hinein, nachdem sie nachgemessen haben, wieviel Raum der Text beansprucht.

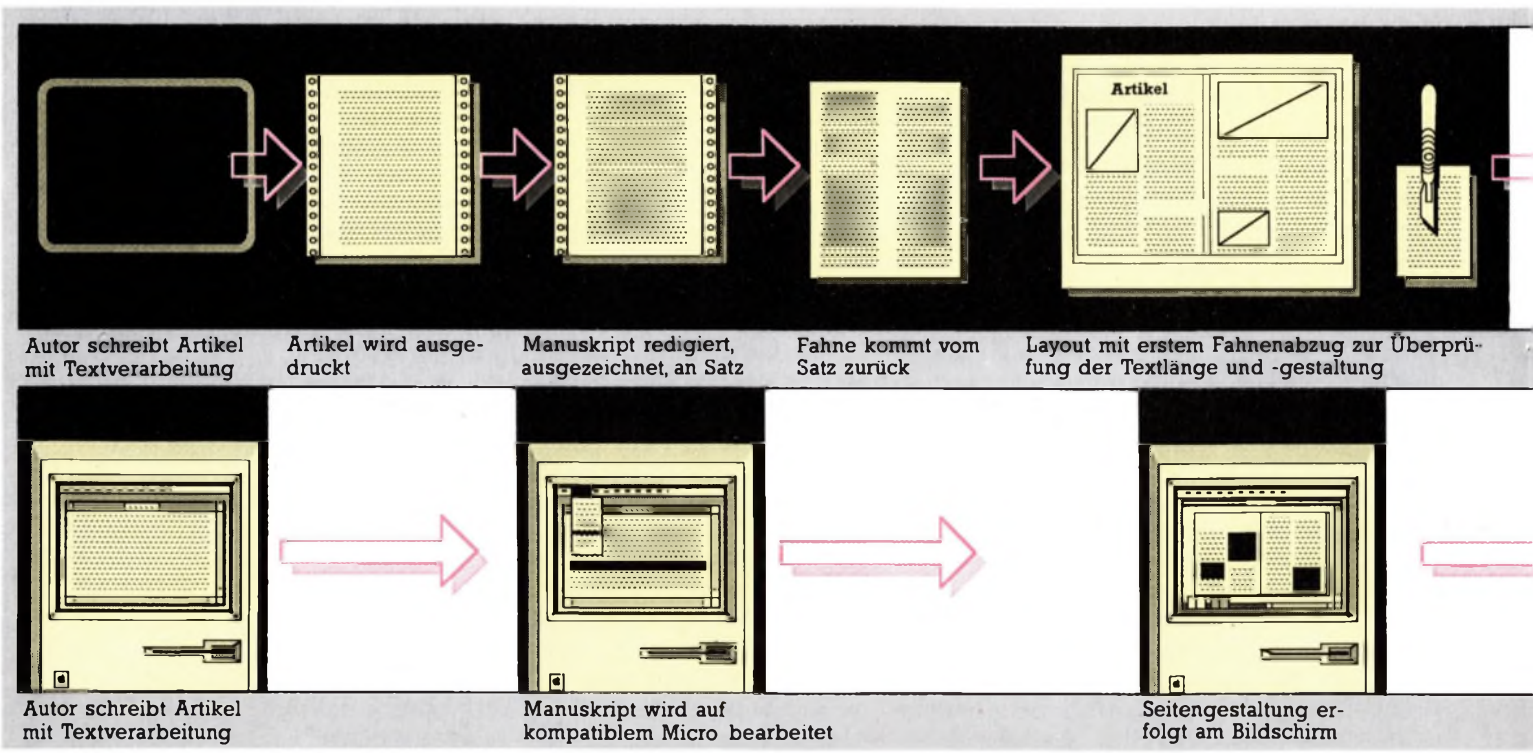
Dabei zeigt sich oft, daß Artikel zu lang oder zu kurz für den vorgesehenen Raum sind. In diesem Fall müssen Redakteur und Produktionsredakteur Kürzungen vornehmen oder Text ergänzen. Kürzungen oder Ergänzungen werden auf der ersten Fahne angebracht, so daß die zweite Fahne bereits die richtige Länge hat. Schließlich wird die Seite mit Diagrammen, Fotos und dem richtigen Text montiert und zum Drucker geschickt.

Es geht auch schneller

Aus dieser Kurzbeschreibung der derzeitigen Magazinproduktion ist zu ersehen, welchen Produktionsprozeß der Beitrag eines Autoren bis zum Fertigprodukt durchläuft. Und wieviel Zeit dabei vergeudet wird. Der Computer des Autoren steht völlig isoliert und ist kaum in den Produktionsprozeß integriert.

Doch es geht auch einfacher, mit einem einer normalen Textverarbeitung nicht unähnlichen Programm. Es enthält zusätzliche Befehle, mit denen etwa unterschiedliche Schriftarten gewählt werden können. Nehmen wir einmal an, die Redaktion benutzt ein Textverarbeitungsprogramm, das mit dem Computer des Autoren kompatibel ist. Der Autor kann eine Diskette abgeben, auf der sein Artikel abgespeichert ist. Ja, er könnte den Beitrag mit einem Modem sogar über Telefon senden. Das Redigieren würde am Bildschirm erfolgen. Selbst wenn daran anschließend der gewohnte Produktionsprozeß einsetzt (Ausdruck des Ma-

Druck auf neuen Wegen



nuskripts und Übergabe an die Setzerei), würde doch der Arbeitsablauf beschleunigt.

Ist die Setzmaschine in der Lage, die Diskette, auf die das Original-Manuskript (redigiert) gebracht wurde, zu lesen, erfolgt seine Weiterverarbeitung lediglich durch einige „magnetische Schritte" – bis zur Herstellung des Satzes.

Man bezeichnet das als „Direkteingabe-System": Der Text wird nur einmal vom Autor eingegeben und dann nacheinander in andere Maschinen gebracht, indem man ihn von Diskette lädt.

Die Möglichkeiten der neuen Technologie indes reichen viel weiter. Neuere Computerentwicklungen, etwa WIMP (Windows (Fenster), Icons (Piktogramme), Mouse (Maus)-Programme) erleichtern dem Layout die Arbeit. Der Layouter kann seine Seite auf dem Bildschirm gestalten, bereits digitalisierte Bilder und Fotos einfügen. Unterschiedliche Schriftgrößen und Verzerrungen werden mit Hilfe der Maus erzeugt. Eine Kopie der Seite wird dann direkt von einem Hochleistungsdrucker ausgegeben.

Tag für Tag

Der Produktionsablauf von Zeitungen unterscheidet sich nur teilweise von dem bei Magazinen. Die schnellen Verarbeitungsanforderungen bei Tageszeitungen in Relation zu ihrer hohen Auflage bedeuten letztlich, daß die neue Technologie diesen Publikationszweig noch viel stärker beeinflussen wird. Die Veränderungen zeichnen sich bereits ab.

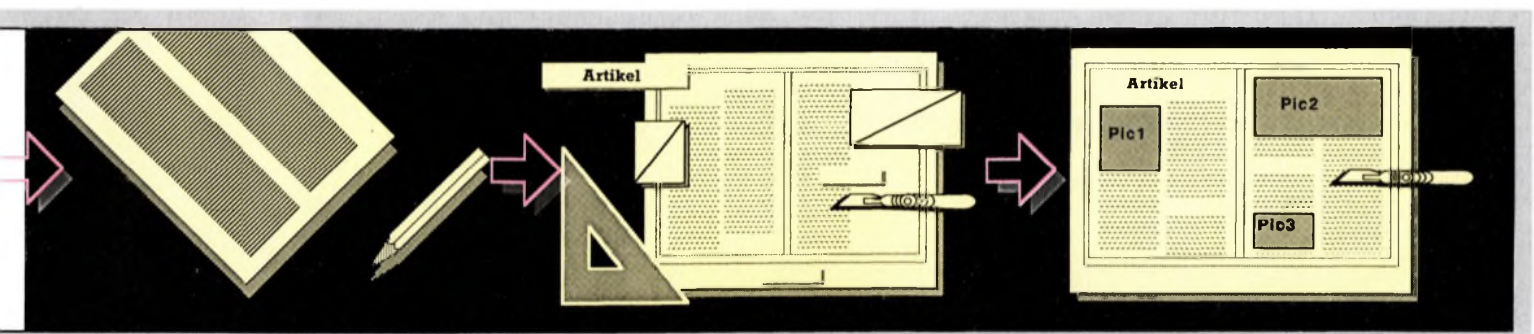
Einige Lokalzeitungen haben als Reaktion auf die zahlreichen kostenlosen Anzeigenblätter, die billig hergestellt werden, ihre Produktionsprozesse computerisiert. Sie kommen mit einem Fünftel der Belegschaft aus, die derzeit erforderlich ist.

Schlagzeile via Modem

Ein wichtiger Aspekt der elektronischen Verlagsverwaltung betrifft den Vertrieb. Überregionale Tageszeitungen und Magazine werden an einem, vielleicht zwei Druckorten hergestellt und dann über Straße und Schiene transportiert und ausgeliefert. Mit elektronisch erzeugten Zeitungen ließe sich dieses Verfahren radikal ändern. Die Herstellung des Papiers mag zwar an einer Stelle erfolgen, der Druck indes kann mittels Datenfernübertragung über gewöhnliche Telefonleitungen an mehreren Orten des Landes erfolgen. Damit werden die Vertriebskosten drastisch reduziert. Zudem ist es leichter, Lokalausgaben herzustellen: Lokalteile werden dem Mantel einfach beigelegt.

Einerseits steckt in der neuen Technik die Gefahr des Arbeitsplatzabbaus. Andererseits können aber auch kleine Zeitungen sich halten, die sonst der Konkurrenz zum Opfer fielen.

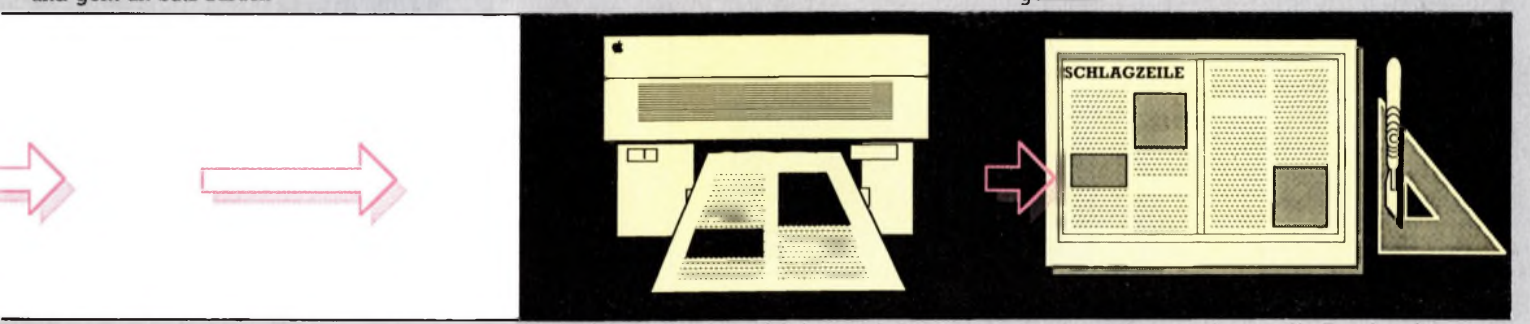
Im zweiten Teil unseres Überblicks wollen wir uns damit befassen, welche elektronischen Herstellungsmethoden auf dem Microcomputermarkt angeboten werden und wie ein Magazin vom Autoren bis zum Fertigprodukt auf einem einzigen Rechner hergestellt werden kann.



Fahne wird korrigiert, auf Länge gebracht und geht an Satz zurück

Korrigierter Satz wird montiert

Fertige Seiten werden an die Druckerei gesandt



Hochleistungsdrucker wirft fertige Seite aus

Fertige Seiten werden an die Druckerei gesandt

Kostenprobleme

Der Einzug von Computern in den englischen Schulunterricht hat eine Reihe von Problemen mit sich gebracht – nicht zuletzt aufgrund der mangelnden Unterstützung seitens der Regierung.

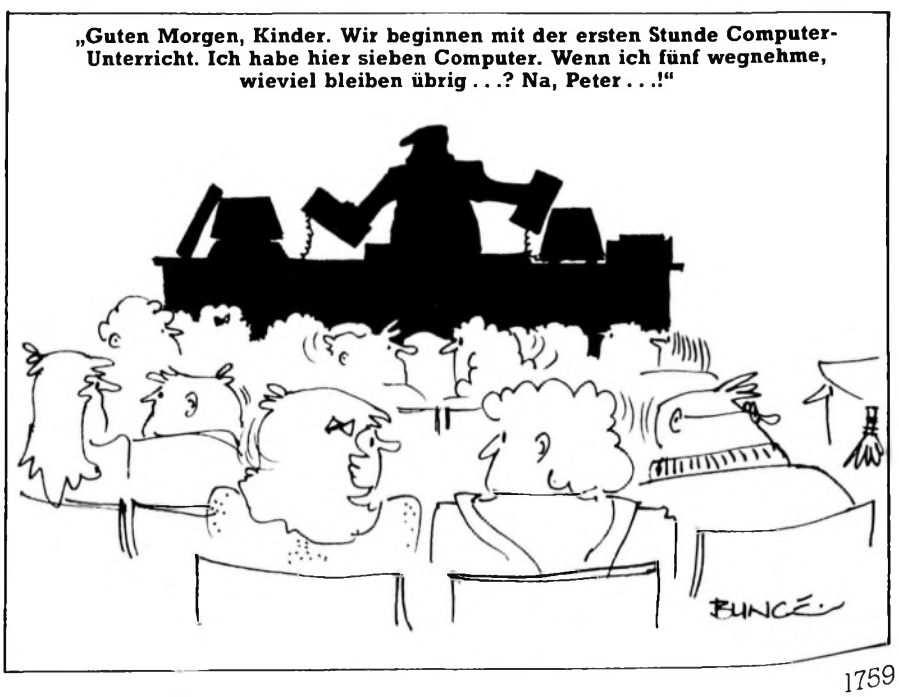
Probleme hat es seit Einführung der Computer in den Unterricht stets gegeben. Sparmaßnahmen, ausgeprägte Ignoranz gegenüber dem Thema an sich und den Schwierigkeiten bei der Ausbildung der Lehrkräfte sind Ursachen für den geringen Einsatz von Computern im Schulunterricht. Die neue Technologie ist von Regierungsbeschlüssen abhängig, und die beträchtlichen Sparmaßnahmen verhindern die angemessene Ausstattung der Schulen. Nach dem Erwerb kostspieliger Hardware ärgern sich viele britische Lehrer darüber, daß keine Mittel für den Kauf geeigneter Software zur Verfügung stehen.

Diese fehlende Unterstützung ist auch Grund dafür, daß mehrere Projekte, an denen Kinder und Computer beteiligt sein sollten, nicht begonnen werden konnten. Man hat kaum darüber nachgedacht, inwieweit Computer in bestehende Lehrpläne einbezogen werden könnten. Folglich gibt es auch keine klare Strategie für ihre Einführung an Schulen. TICCIT und PLATO in den USA sowie das National Development Program in England wurden in der „Vor-Micro-Zeit" Anfang der 70er Jahre durchgeführt, als noch Gelder flüssig gemacht werden konnten. In den 80er Jahren aber sehen sich die Regierungen beider Länder außerstande, ähnliche Projekte zu finanzieren, obgleich ein dringender Bedarf dafür vorhanden ist. An den weiterführenden Schulen wird über BASIC und Computer-Studien generell heiß diskutiert. Aber: Man stößt auf Ablehnung. Grundschulen werden mit einem Computer ausgestattet – der Lehrer muß alleine sehen, wie er damit klarkommt.

LOGO zum Spitzenpreis

Es gibt ernsthafte Kommunikationsstörungen in der pädagogischen Computerwelt. In London investieren viele Schulen in RML-Rechner. Folglich gab das ILECC – das „Inner London Educational Computing Centre" – viel Geld aus und wandte viel Zeit auf, um eine LOGO-Version für RML-Rechner zu schreiben. Inzwischen aber stellte Research Machines Limited selbst eine volle Implementierung der Sprache her. Unabhängig davon warf die ILECC weiter gutes Geld für eine schlechte Sache aus dem Fenster, um das eigene, schlechtere Turtle-Grafik-Programm herzustellen und an den Schulen zu verbreiten.

Weitere Probleme ergeben sich aus den fehlenden Ausbildungsmöglichkeiten für Lehrer. Man hat Computer im Klassenzimmer aufgestellt und erwartete von den Lehrern, daß sie alles aus den Handbüchern lernen würden.



„Guten Morgen, Kinder. Wir beginnen mit der ersten Stunde Computer-Unterricht. Ich habe hier sieben Computer. Wenn ich fünf wegnehme, wieviel bleiben übrig . . .? Na, Peter . . .!"

Unterstützt durch die BBC und von der Regierung für die Benutzung in Schulen empfohlen, spielen Acorns B-Rechner eine entscheidende Rolle im britischen Schulsystem. Die auf diesen Seiten gezeigten Diagramme sind einer Untersuchung entnommen, die BBCs Educational Broadcasting Services Research Unit im Januar 1985 veröffentlichte. Überraschend dabei ist die noch immer währende Präsenz von Commodores Pet und dem Sinclair ZX 81.

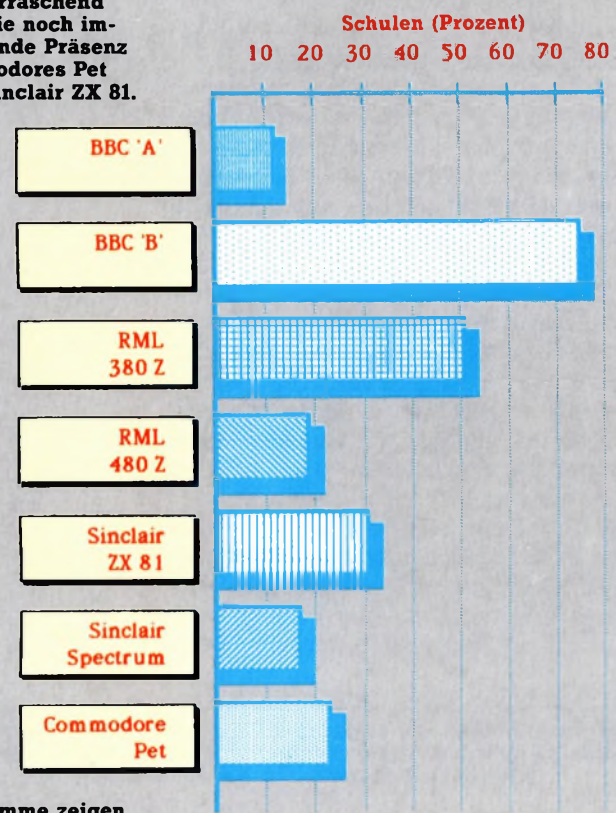
Wegen der Sparmaßnahmen sind Einführungskurse mittlerweile unbezahlbar. Zu jener Zeit, als noch Geld dafür zur Verfügung gestellt wurde, weigerten sich die Schulen, die Lehrer wegen des Unterrichtsausfalls auf Kurse zu schicken.

Diese Situation an Lehrerseminaren ist noch schlimmer. 1983 stellte das Erziehungsministerium besorgt fest, daß es kaum Computerausbildungskurse für Lehrer gäbe. Unterstützt vom Wirtschaftsministerium empfahl man Kurse von mindestens 20 Stunden Dauer. Chris Gregory, Dozent für Mathematik am Bradford College und Mitglied des Projekts MEP (Microcomputers in Education Project), sagte: „20 Stunden sind ohnehin lächerlich, aber viele Kollegen schaffen nicht einmal das." Ein Hauptproblem ist das Fachpersonal. Das Erziehungsministerium beläßt das Personal beim derzeitigen Stand oder streicht Planstellen. Unter diesen Voraussetzungen kann man kaum Computerfachleute an den Schulen finden.

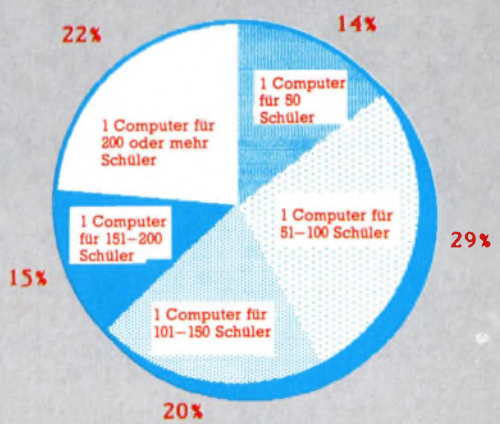
Gute Lernprogramme gelangen nur selten auf den Markt. Die Regierung hat in Hardware, nicht aber in Software investiert. Da die Schulen kein Geld ausgeben können, liegt die Entwicklung von Software im argen. Anläßlich der Educational Publishers Software-Ausstellung 1983 ergab sich, daß die meisten in diesem Bereich tätigen Unternehmen Verluste hinnehmen mußten. Was den Markt noch komplizierter macht, ist der Umstand, daß viele Schulbehörden Geld in eigene Software-Entwicklungen gesteckt haben. Damit sind die teureren kommerziellen Programme noch weniger attraktiv, der Marktanreiz erstirbt.

Sloane Software beispielsweise veröffentlichte kürzlich ein „Hopslide" betiteltes Mathematik-Programm für fünf- bis siebenjährige Kinder. Es ist ein einfaches Programm, das zum Verständnis von Zahlenwert und Zahlenfolge beiträgt. „Hopslide" kostet weniger als so manches Schulbuch und könnte gut verkäuflich sein. Die Inner London Education Authority produziert aber eigene Software. So gibt es ebenfalls ein Hopslide-Programm, das im SMILE-Mathematik-Programmpaket enthalten ist. Es ist identisch mit der Sloane Software-Version, abgesehen davon, daß der Zahlenbereich größer und die Grafik etwas besser ist. Das SMILE-Programm enthält 19 weitere, ähnliche Programme und kostet nur unbedeutend mehr als die Sloane-Version.

Geldmangel und schlechte Organisation sind aber nicht die alleinigen Ursachen für die Misere. Es gibt andere Probleme, die bei der Einführung der neuen Technologie in Schulen hinderlich sind. So hat allein die physische Präsenz von Computern Probleme verursacht. Elektronische Ausrüstung wird in vielen Schulen nachts in einem gesicherten Raum deponiert. Bevor es Computer gab, waren dort zwei oder drei Cassetten- und Videorecorder untergebracht. In einer Grundschule muß nun ein Lehrer zwei- oder dreimal mit Computer, Monitor, Kabeln, Diskettenstation und Software über einen langen Gang gehen, ein paar Treppen steigen, und dies jeden Morgen und Abend. Schließlich weigerten sich Lehrer, die Ausrüstung weiterhin zu schleppen. Die Schulbehörde bestand jedoch darauf, daß die Computer gesichert untergebracht sein müßten. Ergebnis: Die Computer blieben vier Monate un-



Rechnertypen an Schulen



Schüler je Computer

Die Diagramme zeigen deutlich, daß trotz der Versuche der Regierung, das Verhältnis Schüler/Microcomputer an englischen Schulen zu verbessern, eine von fünf Schulen noch immer mit einem Computer für durchschnittlich 200 Kinder auskommen muß.

genutzt im Aufbewahrungsraum, während Lehrergewerkschaft und Behörden versuchten, einen Kompromiß zu finden. Man einigte sich schließlich darauf, Stahlschränke im Klassenzimmer aufzustellen.

Die Situation an vielen Schulen ist beklagenswert. Einfacher Grund dafür: Man hat kein Geld, um Sicherheitssysteme zu installieren. Die Schulen in Englands Ballungszentren werden jährlich mehrfach von Dieben und Einbrechern heimgesucht. Die Versicherungsprämie ist so hoch, daß die Behörden es für preiswerter erachten, gestohlene Gegenstände neu zu erwerben, als sie zu versichern.

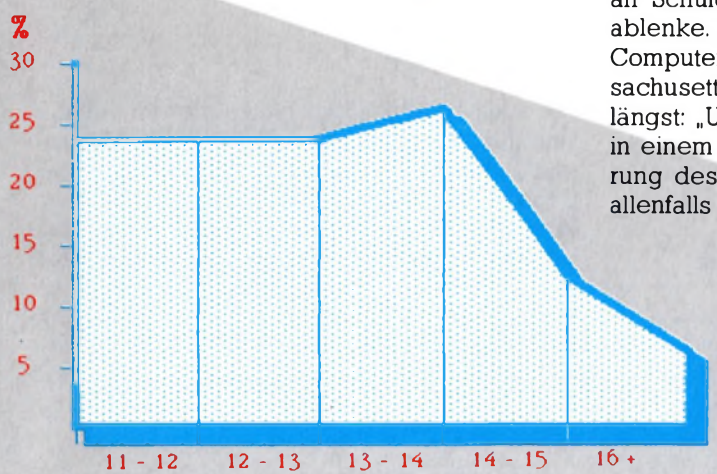
Männersache!

Doch auch, wenn die Computer tatsächlich eingesetzt werden können, gibt es Probleme. Die viel zu wenigen Computer an Schulen scheinen überdies im Unterricht Jungen vorbehalten zu sein. Jungen sind auch meist Besitzer von Heimcomputern, die sie hauptsächlich zum Spielen benutzen. Ihre Games sind oft militant und aggressiv. Im Klassenzimmer setzt sich der gewohnte Umgang mit dem Rechner fort. Der Computer wird als männliche Domäne betrachtet. Die Lernprogramme sind meist für Jungen geschrieben, und diese monopolisieren die Computer zu Lasten der Mädchen. Ein Sprecher der Equal Opportunities Unit der Inner London Education Authority sagte: „Wir sind uns dieser Unausgewogenheit beim Computerunterricht bewußt. In reinen Mädchen-Schulen gibt es das Problem nicht. Es existiert nur in gemischten Schulen, wo Jungen dominieren. Dort müssen wir die Mädchen dazu ermutigen, Interesse am Computer zu finden, um so das Gleichgewicht herzustellen." Interessant ist das Forschungsergebnis der Universität Edinburgh, die den Einsatz von LOGO an Schulen untersuchte: Mädchen wie Jungen waren gleichermaßen interessiert.

Das größte Problem ist jedoch, wie die Regierung Ausbildung bewertet, nämlich nach dem Produktivitäts- und Kostenschlüssel. Betrachtet man den Computer als pädagogisches Lernwerkzeug, ist die Zukunft erschreckend: Kinder werden zum Bestandteil einer computerisierten Produktionsfolge, Lehrer werden durch Rechner ersetzt und Lehrpläne von Regierungskomitees festgelegt. Wie Seymour Papert anmerkte: „Es gibt starke Kräfte in der Welt, die auf eine starrere Gesellschaft drängen und den Computer auf eine Weise nutzen wollen, die das Leben weniger aufregend macht, starrer und unsozialer. Was können wir tun, um ihnen zu begegnen . . .? Wir können sagen: ‚Verbrennt die Computer!'. Aber das wäre Zeitvergeudung . . . Oder wir können versuchen, das Ruder herumzureißen und den Computer auf eine sozial wünschenswerte Weise zu nutzen."

Die Ausbildung auf beiden Seiten des Atlantik steckt in der Krise. Mangel an Investitionen, niedrige Löhne sowie eine generelle Abkehr von sozialen Errungenschaften verlangen ihren Tribut. Es gibt Menschen, die damit argumentieren, daß die Einführung von Computern an Schulen von den eigentlichen Problemen ablenke. Joseph Weizenbaum, Professor für Computerwissenschaft an der Universität Massachusetts (Institute of Technology) sagte unlängst: „Unsere Schulen befinden sich bereits in einem hoffnungslosen Zustand. Die Einführung des Computers zu diesem Zeitpunkt ist allenfalls eine Ablenkung . . . Er wird benutzt . . . um einen technologischen Fixpunkt zu setzen . . . und dabei fundamentale Probleme zu kaschieren, indem die Illusion erzeugt wird, man setze sich mit ihnen auseinander."

Jener Teil der Gesellschaft, der letztendlich über Einsatz oder Nicht-Einsatz von Computern entscheidet, sind die Kinder. Sie sagen immer ihre Meinung und wissen oft genau, ob etwas gut für sie ist oder langweilig und unwichtig. Sie reagieren auf gute Software. In einer vom Computer bestimmen Umwelt schaffen Kinder ihre eigene Computer-Kultur und entwickeln ihre eigene Sprache.



Die Grafik zeigt die „Computererfahrung", die Schulkinder in England haben. Dabei werden die Auswirkungen der Spezialisierung im oberen Altersbereich deutlich. Ältere Schüler finden in ihren Leistungsfächern oft keine Anwendungsmöglichkeit für Computer. Lediglich 30 % der befragten Altersgruppen geben an, den Computer „im Griff" zu haben.

Computer „im Griff"

Herzspezialisten

Innerhalb der letzten Jahre hat sich Motorola Incorporated aus einer bescheidenen Außenseiterposition an die Spitze der Hersteller microelektronischer Bauteile vorgearbeitet.

Wie viele andere erfolgreiche Konzerne hat sich auch Motorola aus einem Ein-Mann-Unternehmen entwickelt: Paul Galvin gründete 1928 in Chikago die „Galvin Manufacturing Corporation", die auf die Produktion von Radioapparaten spezialisiert war. In den 30er Jahren wurde das Programm um Autoradios und Funkgeräte für die Polizei erweitert. Bereits in den 40er Jahren begann die Firma – jetzt als Motorola Incorporated – mit der Halbleiterproduktion.

1959 starb Paul Galvin, neuer Chef wurde sein Sohn Robert. In den folgenden Jahren geriet Motorola im Halbleiterbereich und bei der Unterhaltungselektronik unter starken Konkurrenzdruck. Die Wirtschaftskrise Mitte der 70er Jahre steigerte die Verluste der Gesellschaft noch weiter – eine Umorientierung wurde unvermeidlich. Neue Mitarbeiter wurden angeworben, darunter viele aus dem Hause des Erzrivalen Texas Instruments. Motorolas neuer Weg führte fort von den traditionellen Bereichen der Elektronik, auf denen das Unternehmen nicht mehr konkurrenzfähig war. Das Ziel hieß jetzt High-Tech-Microelektronik.

Gewagte Operationen

Um die riesigen Summen für Forschung und Entwicklung aufzubringen, verkaufte Motorola Teile der Firma. Im Gegenzug wurden Unternehmen aus Bereichen übernommen, in die Motorola vordringen wollte – ein riskantes Vorgehen, aber es gab keine andere Wahl.

Offensichtlich hat sich das Wagnis gelohnt. Zwar stand Motorola auch im zweiten Teil der 70er noch im Schatten anderer Unternehmen der Halbleiterbranche, heute ist das Ziel jedoch erreicht – der einstige Marktführer Texas Instruments wurde überrundet. Robert Galvin sagt dazu: „Die früheren Konkurrenten von Motorola existieren einfach nicht mehr. Sie haben sich nicht rechtzeitig den veränderten Gegebenheiten angepaßt."

Bis heute hat man bei Motorola allerdings Schwierigkeiten dabei, Produkte zum richtigen Zeitpunkt auf den Markt zu werfen. Mitte der 70er Jahre hatte der Motorola-Chip 6800 nicht nur den im erfolgreichen Apple-Heimcomputer eingebauten Prozessor Mostek 6502 als Konkurrenten, sondern auch noch den Intel 8085 und den Zilog Z80 für CP/M-Rechner. 1976 kam der Motorola 6809 heraus, damals der anerkannt beste Acht-Bit-Prozessor. Der Wettbewerb um den Massenmarkt war jedoch bereits verloren – der Chip wurde nur in wenigen Rechnern als „Herz" eingesetzt, etwa dem Tandy Color Computer und dem Dragon.

Der Schrittmacher

Trotzdem investiert die Gesellschaft nach wie vor kräftig in die Forschung. „Maximalen Fortschritt zum frühestmöglichen Zeitpunkt" heißt die Devise von Robert Galvin. Im Markt der 16-Bit-Prozessoren hat Motorola denn auch eine bessere Position: Der 68000-Microprozessor erschien bereits 1979. Heute sitzt dieser Chip in Apples Lisa und auch im Macintosh, im Atari ST und im Amiga. Sinclair hat ihn für den QL übernommen. Der Chip ist außerordentlich leistungsfähig, verfügt über 17 32-Bit-Register, einen 16-Bit-Datenbus und einen 24-Bit-Adreßbus, damit genau über das, was der Markt verlangt, auf dem heute Massen umgesetzt werden können, auf dem Gewinne erzielt werden.

In den Forschungszentren von Motorola in Phoenix (Arizona), Genf (Schweiz) und East Kilbridge (Schottland) werden weiterhin neue Produkte entwickelt. Die schottische Fabrik erzeugt MOS- und CMOS-Bauteile für einen breitgefächerten Anwendungsbereich. Heute ist Motorola in fünf Unternehmensbereiche aufgeteilt (Kommunikation, Halbleiter, Informationssysteme, Industrieelektronik und Militärtechnik).

Motorola-Chef Robert Galvin

Motorola-Zentrale in Illinois, USA

Der Standard

MS-DOS von Microsoft hat sich zum Standard für 16-Bit-Maschinen entwickelt und bildet die Grundlage für das vielkopierte IBM PC-DOS.

In der Anfangstagen der Acht-Bit-Micros waren der Z80 von Zilog in Verbindung mit dem Betriebssystem CP/M Marktführer. Gary Kildall – der Gründer von Digital Research – hatte aber das Betriebssystem CP/M ursprünglich für den Intel 8080 entwickelt.
Als sich IBM dem Microcomputer widmete, neigte sich die Ära der Acht-Bit-Systeme zwar ihrem Ende zu, doch gab es noch keine Kombination von CPU und Betriebssystem, die die Nachfolge von Z80 und CP/M angetreten hatte. Für IBM bestand daher die Möglichkeit, mit ihren 16-Bit-Maschinen eine völlig neue Hard- und Softwarekombination einzuführen. Mehrere Gründe sprachen jedoch dagegen:
1) Trotz beherrschender Stellung in der Groß-EDV besaß IBM keine Erfahrung in der völlig andersartigen Welt der Microcomputer.
2) Der Erfolg von DEC (Digital Equipment Corporation) im Bereich der Minicomputer hatte IBM gezeigt, daß Anwender nicht mehr bereit waren, IBM-Produkte blind zu kaufen.
3) Ein völlig neues System hätte viel Zeit für die Entwicklung einer Softwarebasis gebraucht. Ohne Unterstützung vieler unabhängiger Softwarehäuser hätte sich IBM in der sich schnell verändernden Microcomputerwelt nur schwer behaupten können.
4) Hätte IBM Hard- und Software selbst entwikkelt, wäre das Endprodukt für Anwender nicht mehr erschwinglich gewesen.
IBM entschied sich daher, das „Beste" der bekannteren Betriebssysteme zu nehmen und es an seine 16-Bit-Umgebung anzupassen.

Nichts Neues im Vergleich

Das Ergebnis dieses konservativen Konzeptes war ein recht langsamer IBM PC, der im Vergleich mit Maschinen der gleichen Generation nichts Neues bot. Selbst die Wahl des Hauptprozessors Intel 8088 und die damit verbundene Entscheidung für die Chipfamilie des 8086 (8086/186/286/386) hatte rein praktische Gründe. Der 8088 arbeitete intern mit einer 16-Bit Struktur, besaß aber einen Acht-Bit externen Datenbus und konnte so die bewährten (und billigen) Peripheriechips im Acht-Bit-Format einsetzen. Die Anlage lief langsam, da jedes 16-Bit „Wort" in zwei Teilen geladen werden mußte.

Inkompatibel

**Hauptursache erster Kompatibilitätsprobleme des IBM PC war die Verlegung der Sprungtabelle um zwei Speicherseiten nach „unten". Die Tabelle befand sich nun bei der ROM Adresse 400H (die MS-DOS Tabelle beginnt bei 600H). Dieser Teil des BIOS (Basic Input/Output System) wurde von IBM urheberrechtlich geschützt und stellte Firmen vor Probleme, die hundertprozentig kompatible Nachbauten des IBM PC herstellen wollten. Dennoch lief Software, die die Hardware über die korrekten Systemaufrufe des MS-DOS ansprach, auf allen MS-DOS Geräten.
Diese Kompatibilität hat jedoch ihren Preis. Die Systemaufrufe verlangsamen die Ausführgeschwindigkeit und verhindern den Einsatz späterer Hardwareverbesserungen. Aus diesen Gründen entwickelten etliche Softwarehäuser hard- (oder firmware-) abhängige Programme, die nur in einer dem IBM PC sehr ähnlichen Hardwareumgebung liefen. Einige Softwarehersteller machten ihre Produkte sogar absichtlich inkompatibel. So ändert Digital Research beispielsweise bei GEM-Anwendungen einige wenige Codebytes, die für jeden Maschinentyp den Kauf einer anderen Version nötig machen – selbst bei hundertprozentig kompatiblen Nachbauten ist dies notwendig.**

„Quick and Dirty"

Für Betriebssystem und Software kontaktete IBM die Firma Microsoft, die hauptsächlich für ihren leistungsfähigen BASIC-Dialekt bekannt war. Microsoft hatte zu diesem Zeitpunkt Kontakt zu „Seattle Computer Products", die auf der Basis von Intel-Chips 16-Bit-Systeme entwikkelte. Der Programmierer Tim Patterson hatte für die 8086-Familie ein Maschinencode DOS (SCP-DOS) geschrieben, das sich eng an CP/M lehnte. Sein System wurde auch QDOS (nicht zu verwechseln mit dem Sinclair OS für den QL) genannt, wobei die Abkürzung „Quick and Dirty Operating System" – „schnell und unsauber" geschriebenes OS – bedeutete. Die Beschreibung traf zu: Pattersons OS funktionierte zwar, war aber nicht durchkonstruiert und hatte Fehler. Microsoft kaufte nun die Rechte für dieses System und gab es in Lizenz für Original Equipment Manufacturers (OEM) – Hersteller von Geräten, die Bauteile anderer Hersteller enthalten – heraus.

Aus der MS-DOS Version (Version 1) entwikkelte IBM für den PC ein eigenes System: PC-DOS. Bei jeder wichtigen Weiterentwicklung von MS-DOS erschien eine entsprechende PC-

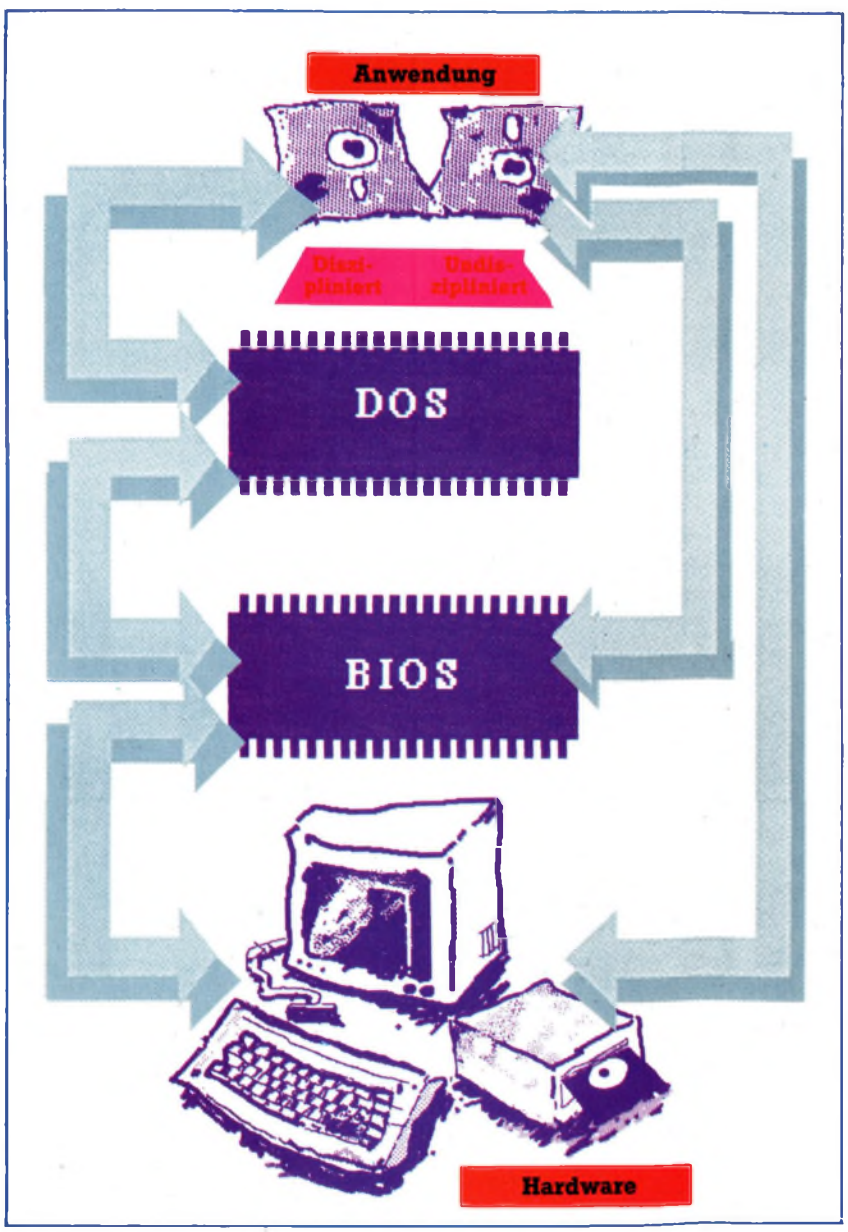
Die Vormachtstellung des IBM PC und seine MS-DOS Variante (mit Namen PC-DOS) sicherte MS-DOS den ersten Platz unter den 16-Bit-Betriebssystemen. Das System hatte seinen Anfang in einer recht groben CP/M-Anpassung eines früheren Angestellten von Digital Research.

Diszipliniert
Software, die mit standardmäßigen MS-DOS Aufrufen arbeitet, wird „diszipliniert" genannt. Das DOS spricht die BIOS-Routinen dabei indirekt über eine Sprungtabelle an. Unser Bild zeigt, wie diese Art der Programmierung aufgebaut ist. Programme, die mit absoluten BIOS-Adressen arbeiten oder MS-DOS völlig umgehen und die Hardware direkt ansprechen, heißen „undiszipliniert". Auf dem IBM PC liegt das BIOS in einem urheberrechtlich geschützten ROM. Die Sprungtabelle fängt bei der Adresse 400H an und liegt damit 512 Bytes (zwei Hexadezimalseiten) unter dem Anfang.

DOS Version, so daß jede Aussage dieser Serie über MS-DOS auch auf PC-DOS zutrifft. Unterschiede zwischen beiden Systemen fallen für Anwender kaum ins Gewicht. Intern gibt es einige Abweichungen, und auch die Numerierung der Disketten variiert etwas. Theoretisch wären dies alle Unterschiede, doch leider adressieren viele Anwendungsprogramme des IBM PC die Hardware direkt und nicht über die entsprechenden Systemroutinen. So findet sich hier die Hauptursache für Kompatibilitätsprobleme (und nicht im Unterschied zwischen PC-DOS und MS-DOS). Programme, die das DOS korrekt ansprechen, sollten ohne Anpassung unter jedem MS-DOS oder PC-DOS der gleichen (oder einer späteren) Version laufen. Selbstverständlich funktionieren DOS-Verbesserungen, die erst kürzlich (z. B. in Version 3.1) herausgegeben wurden, nicht mit früheren Versionen ohne diese Fähigkeiten.

Die erste MS-DOS-Version war eigentlich ein an die 16-Bit-Familie von Intel angepaßtes CP/M-80, das in vielen Befehlen und Abläufen noch CP/M ähnelte. Tim Patterson hatte jedoch einige Befehle bedienerfreundlicher gestaltet.

Für Verbesserungen des Systems orientierte sich Microsoft zunächst an Xenix, ihrer eigenen Version von Unix (das Mehrplatz-Betriebssystem von Bell Labs) und versuchte, über gemeinsame Systemaufrufe einen gewissen Grad an Kompatibilität herzustellen. Für den Anwender hatten jedoch in DOS 2 die E/A-Richtungsumkehrung, die „Pipelines" und das hierarchische Inhaltsverzeichnis weit größere Bedeutung. All diese Konzepte gingen auf das wohlbekannte Unix zurück.



Mehrplatzanwendungen

DOS 3 – die neueste MS-DOS-Version – enthält alle Eigenschaften des DOS 2, unterstützt darüberhinaus noch Mehrplatzanwendungen. OEMs für Netzwerksysteme (z. B. Apricot oder Research Machines Limited) arbeiten mit Version 3.1 – einer von beiden Hauptversionen dieser Generation – und dem darin eingebauten Microsoft Netzwerk.

Da jedoch das Betriebssystem dadurch sehr umfangreich wird, steht für andere Umgebungen das von allen Netzwerkfähigkeiten des 3.1 befreite DOS 3.0 zur Verfügung, das aber auch den Mehrfachzugriff auf Dateien unterstützt. Diese Softwarelösung läßt sich gut für Mehrplatzanwendungen einsetzen, die mit gleichzeitigem Zugriff auf gemeinsame Datenbanken arbeiten.

Der Intel 80286 des IBM PC-AT hat einen Adreßraum von drei MBytes und Hardwareschaltungen zur Sperrung von Dateibereichen. Auf dem 8086 und 80186 ist es unter DOS 3.0 zwar möglich, den Zugang zu einem Bereich mit zusätzlichen Systemaufrufen zu sperren, doch geht dies zu Lasten der Verarbeitungsgeschwindigkeit. Durch weitere Verbesserungen sind inzwischen fast alle Fehler des alten „QDOS" ausgemerzt, und MS-DOS stellt ein ausgereiftes Produkt dar.

Wichtige Verbesserungen

Eine der wichtigsten Verbesserungen ist Microsoft Windows. Wie GEM von DR bietet MS-Windows eine WIMP-ähnliche Umgebung mit Multitasking und einfachem Datenaustausch zwischen den Anwendungen. Das System wird als Erweiterung von MS-DOS/PC-DOS eingesetzt und soll auf den herkömmlichen Computern eine neue Ära der Bedienerfreundlichkeit einleiten. In der nächsten Folge untersuchen wir die Grundbefehle von MS-DOS. Vermutlich am häufigsten wird wohl dabei auf allen Systemen der Befehl „Directory" eingesetzt. Er zeigt nämlich alle Dateinamen einer Diskette, oder – von DOS 2 an aufwärts – ein „Directory", einem Teil der Diskette.

Jobs für Hacker?

Ihre Computer-Kenntnisse bringen Sie vielleicht auf die Idee, auch Ihren Broterwerb im Computer-Bereich zu suchen. Die Rechner-Industrie bietet mehr Stellen an als andere Berufszweige.

An erster Stelle bei einer Entscheidung für das Computer-Business muß immer die Frage stehen, welchen Teilbereich aus diesem komplexen Arbeitsfeld Sie wählen möchten: Superjobs liegen auch hier nicht auf der Straße. Karrieremöglichkeiten gibt es zwar, die einzelnen Laufbahnen unterscheiden sich aber sehr stark. Der eine Beruf bietet mehr Gehalt, der zweite interessante Technologieerfahrungen, ein dritter vielleicht gute Zukunftschancen oder ein besonders fruchtbares Arbeitsklima . . .

Bedenken Sie, daß eine Computer-Karriere auch zur Einbahnstraße werden kann. Der Einstieg legt Ihre künftigen Entwicklungsmöglichkeiten fest, besonders dann, wenn der Arbeitgeber in Ihre Ausbildung investiert oder Sie ein ganz spezielles Fachgebiet auswählen.

Die Arbeit mit Computern ist außerordentlich vielschichtig, aber auch hier kommen die üblichen Abstufungen vor. Jeder Job ist mit unterschiedlichen Anforderungen verbunden, wird unterschiedlich bezahlt und hat unterschiedlich gute Karrierechancen. In der Produktion von elektronischen Geräten oder Computern sind verschiedene Kategorien von Angestellten mit der Hardware beschäftigt: Fließbandarbeit gibt es hier genauso wie Tätigkeiten als Produktionsingenieur, Entwickler oder gar als Forscher.

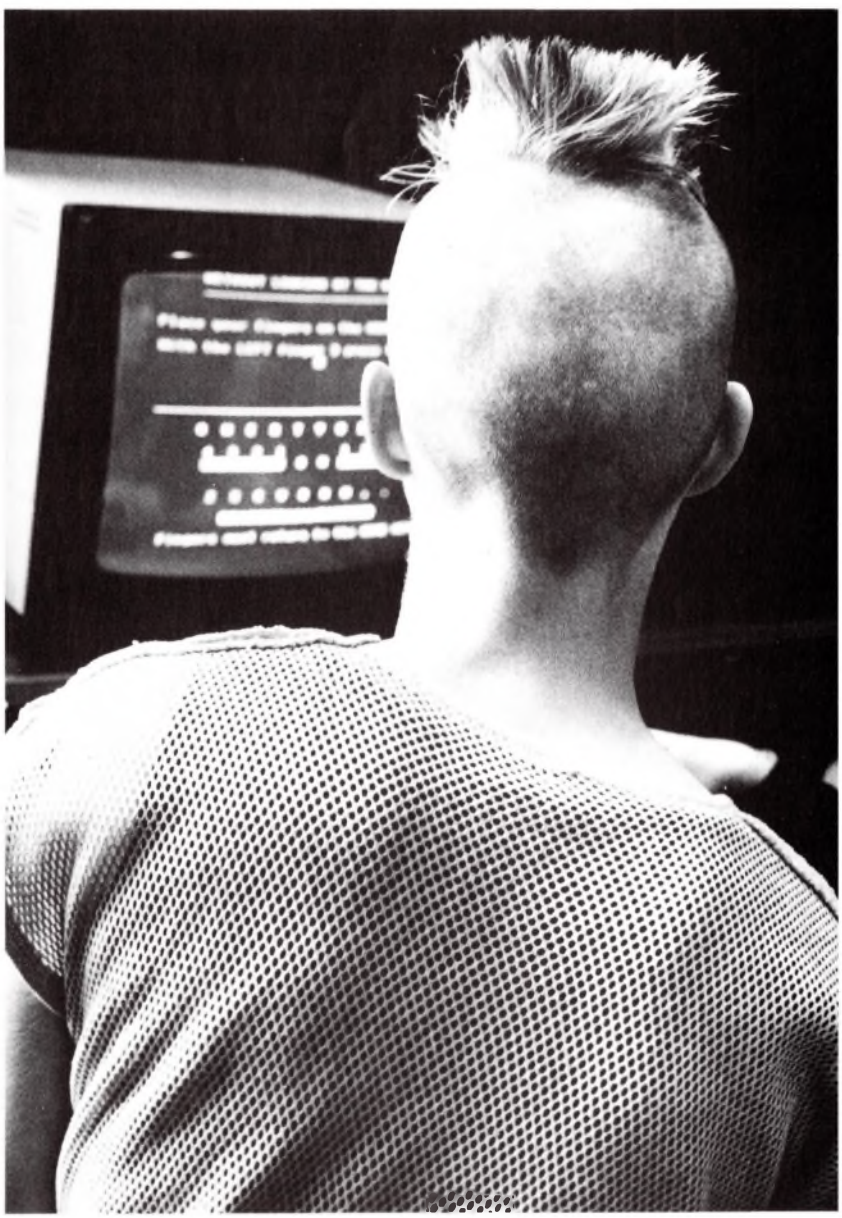
Fachmann gesucht

Systementwicklung – ob Microprozessor oder Großrechner – leistet nur der erfahrene oder überaus begabte Fachmann, wobei die Ingenieurlaufbahn auch mit eher durchschnittlichem Talent gewählt werden darf. Erfolgreiche Firmen stellen zunehmend mehr Personal für die Entwicklung neuer Produktgenerationen ein, um Anschluß zu halten.

Jobs in der Entwicklung werden ausschließlich an Studienabsolventen vergeben, die gute Bezahlung, Sicherheit und Prestige erwarten können. Akademiker überwiegen gegenüber Nichtakademikern im industriellen Bereich zahlenmäßig um ein Vielfaches. Als Studienfach empfehlen sich für dieses Gebiet die Elektrotechnik oder das Informatikstudium an einer Fachhochschule oder Universität. Die drei- bis fünfjährige Ausbildung vermittelt die theoretischen Grundlagen und fortgeschrittene praktische Anwendung.

Als Alternative dazu gibt es auch noch den Berufsweg des staatlich geprüften Technikers, der im allgemeinen als Weiterbildung nach einer Lehre eingeschlagen wird. Gute Schulzeugnisse allein genügen selten. Im allgemeinen wird erwartet, daß ein neuer Mitarbeiter Schaltungen verstehen kann und die Grundprinzipien eines Microprozessors kennt.

Auch den findigsten Programmierer oder Hobby-Elektroniker wird man nicht gleich zu Anfang an den Entwurf von PCs oder Chips



Im Computer-Bereich finden sich noch offene Stellen. Heimcomputer-Erfahrung reicht hier jedoch selten aus. Vorsicht ist bei „Schmalspur"-Ausbildungen geboten – gleichgültig, ob sie von einer Schule oder als Pseudo-Ausbildung in einem Betrieb offeriert werden.

Computer-Einzelhandel

Eine große Zahl jüngerer Leute drängt in eine Tätigkeit als Verkäufer im Computer-Fachhandel. Der Verkauf über den Ladentisch an den Endverbraucher kann schnell seinen Glanz verlieren. Oft ist die Bezahlung schlecht, die Umsätze sind unregelmäßig. Es ist fraglich, ob der Markt für den kleinen Händler auf die Dauer tragfähig bleibt. Allerdings kommt es auch vor, daß sich Mini-Firmen zu leistungsfähigen Handelsunternehmen entwickeln.

Für den Neuling bietet der Verkauf die Möglichkeit, viel über PCs zu erfahren. Gründliche Einarbeitung in verschiedene Systeme und Programmiertechniken ist aber die Ausnahme – dafür fehlt fast immer die Zeit. Trotzdem kann der Verkauf als Vorbereitung für die Bewerbung bei größeren Firmen genutzt werden.

setzen. Davor steht das Training anderer wichtiger Fähigkeiten. Ein Entwicklungsprojekt wird normalerweise auf ein Team von Dutzenden von Angestellten verteilt.

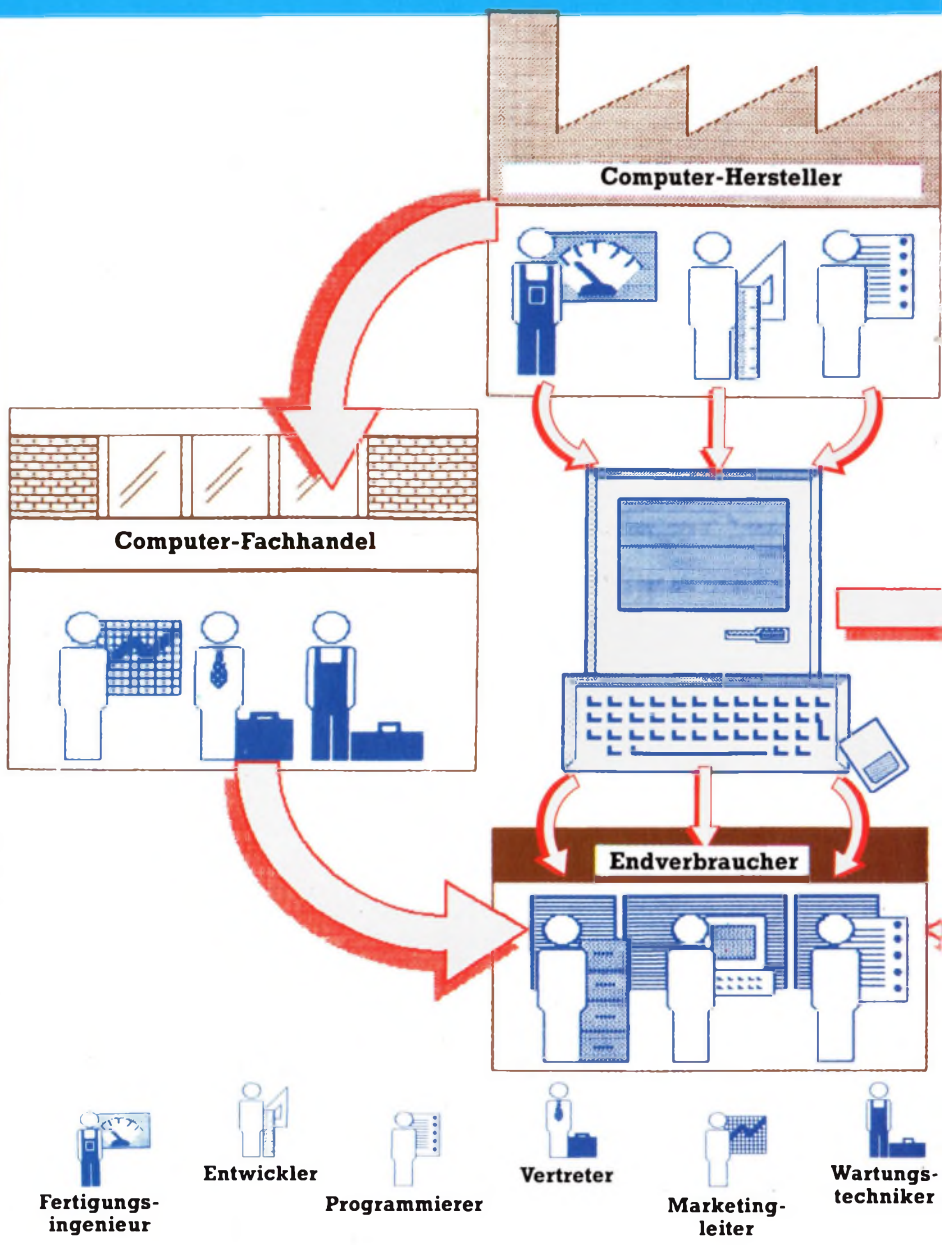
Man neigt dazu, gerade im Computer-Business und in der Elektrotechnik der Fähigkeit zum wissenschaftlichen Arbeiten einen hohen Stellenwert einzuräumen. Die schnellen Fortschritte dieses Arbeitsbereiches machen Fortbildung und ein analytisches Vorgehen bei der Problemlösung unabdingbar. Leider hält sich bis heute die Meinung, daß Frauen sich schnellem Umdenken verschließen. Der prozentuale Anteil weiblicher Mitarbeiter in der Elektronik ist nach wie vor verschwindend gering, beginnt allerdings langsam zu wachsen.

Auch heute noch kann es gelegentlich vorkommen, daß Begeisterung eine formale Ausbildung ersetzt. Findiges Wartungspersonal wird immer gebraucht, aber die Aufstiegsmöglichkeiten und auch die Bezahlung sind wenig reizvoll. Offene Stellen finden sich meist nur bei kleineren Microcomputer-Firmen.

Die meisten Arbeitsplätze gibt es in der Software-Entwicklung und in der Wartung. Programmierer stellen fast die Hälfte aller Angestellten im Computerbereich. Ein weiteres Drittel entfällt auf den Vertrieb.

Die meisten Programmierer arbeiten noch in der reinen Datenverarbeitung (DV), der mehr und mehr dem Bereich „Management Information Service" (MIS) zur Konkurrenz wird. Viele Programmierer sind auch für Computerhersteller, Softwarehäuser oder Computer-Dienstleitungsunternehmen tätig.

Eine Tätigkeit in der Datenverarbeitung ist sehr oft der erste Schritt zum Geschäft mit dem Computer. Allerdings nimmt die Wichtigkeit dieses Bereiches ab, weil Computer immer leichter zu bedienen sind und zunehmend selbsttätig arbeiten können. Eine Stellung als Operator, Programmierer oder System-Analytiker kann jedoch Sprungbrett sein.



Die Programmierung ist heute ein hochspezialisiertes Geschäft. Ein Fachmann in den Sprachen FORTH oder PASCAL wird nicht sofort auf die Feinheiten von COBOL oder FORTRAN umschalten können. Zusätzlich verkompliziert sich die Situation durch das Übergewicht von PASCAL und COBOL im kommerziellen Bereich gegenüber FORTH und FORTRAN

Sprünge beim Gehalt

Tätigkeit	September 84	März 85
Computer-Manager	6,6	8,9
Leiter DV	9,6	8,6
Leit. System-Analytiker	9,0	10,5
Leitender Analytiker	7,3	7,9
Leitender Programmierer	4,8	8,6
Programmierer (Analytik.)	6,5	7,8
Programmierer	5,7	1,6
Alle Beschäftigen	**6,6**	**7,6**

Die Veränderungen in der Gehaltsstruktur sind im Computer-Bereich immer wieder für eine Überraschung gut: Mehr als in anderen Industriezweigen schlagen hier Marktverschiebungen sofort auf das Gehaltsniveau durch. Die nebenstehende Statistik stammt aus Großbritannien und zeigt den durchschnittlichen Gehaltszuwachs für zwei ausgewählte Monate. Die Statistik ist allerdings nicht inflationsbereinigt – sie zeigt, daß auch im aufstrebenden Computergeschäft bestimmte Berufsgruppen, etwa die Programmierer, die Möglichkeit eines Reallohnabbaus einkalkulieren müssen.

bei wissenschaftlich/technischer Anwendung.

Ein nicht unerheblicher Teil – etwa ein Drittel – aller im Computerbereich Tätigen arbeitet für Herstellerfirmen. Außer in der Produktion selbst, der Geräteentwicklung und Wartung beschäftigen Computerfirmen eine große Zahl von Vertriebsleuten. Der Vertrieb wird entweder vom Hersteller selbst, von einer unabhängigen Vertriebsfirma oder vom Fachhandel übernommen; gelegentlich beteiligen sich auch Softwarehäuser daran.

Wie in anderen Industriezweigen werden auch Computer-Vertriebsleute in drei Kategorien eingeteilt. Zum einen gibt es Direktvertreiber, obwohl der „freie" Verkauf in diesem Bereich zu den Ausnahmen zählt. Im allgemeinen wird dem Vertriebsmann ein gewisser Kundenstamm zugewiesen.

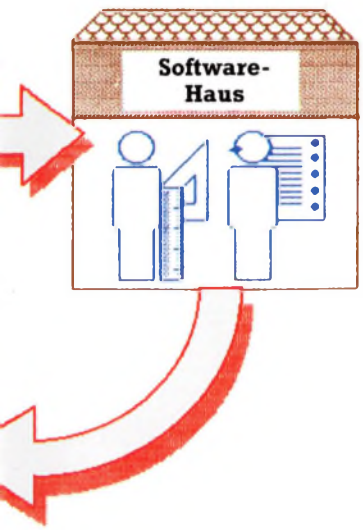
Eine zweite Kategorie sind die Marketing-Fachleute. Sie sollen (speziell im Großrechner-Bereich) die Beziehungen zu Anwendern pflegen, die sich bereits für ein System entschieden haben. Gelegentlich werden auch Neukunden mitbetreut. Großfirmen wie etwa IBM bilden ihre Verkäufer als Gesprächspartner und Helfer der DV-Leiter in den Kundenfirmen aus. In kleineren Firmen hat der Vertriebsdirektor die Aufgabe, dafür zu sorgen, daß Fach- und Großhändler die Produkte der eigenen Firmen bestellen und verfügbar halten.

Die dritte Vertriebsgruppe ist für das Zusammenwirken aller Komponenten – Hardware, Software und System – zuständig. In dieser Position findet man viele ehemalige Programmierer oder Ingenieure.

Die Wahl einer geeigneten Ausbildung ist nicht ganz einfach. Bevor Sie mehr oder weniger zufällig in irgendeinen Kursus hineinstolpern, müssen Sie sich unbedingt genau informieren. In den letzten Jahren schießen – auch als Folge der hohen Arbeitslosigkeit – Computerschulen wie Pilze aus dem Boden. Aber es ist Vorsicht angebracht: Nicht jede Kurzausbildung ist seriös und vermittelt wirklich fundiertes Wissen.

Eine weitere Möglichkeit der Ausbildung neben dem Beruf wird von Fachhochschulen angeboten, etwa das Studium der Elektrotechnik oder Informatik in Abendkursen. Man darf die Doppelbelastung aber nicht unterschätzen. Besonders Kandidaten mit Familie werden hierbei auf eine harte Probe gestellt. Das erworbene Diplom ist allerdings dem normalen Fachhochschuldiplom gleichwertig und deutet auf ein großes Standvermögen hin.

Die Ausbildung an Universitäten und Fachhochschulen wird häufig auf der Industrieseite mit Mißtrauen betrachtet. Den staatlichen Einrichtungen sagt man mangelnde Flexibilität nach. Viele Lehrinhalte seien bereits überholt, das Computerzeitalter an den Bildungseinrichtungen vorbeigegangen. Auch das ist zu berücksichtigen: Es gibt gerade an den Fachhochschulen und Universitäten gravierende Unterschiede in der Qualität und Aktualität. Aber auch überbewerten darf man diesen Faktor nicht. Die Fähigkeit zum wissenschaftlichen Arbeiten, zum Umgang mit Fachliteratur und ähnliches sind vom konkret vermittelten Stoff unabhängig und können in der Praxis die wertvollere Hilfe sein.



Zwar ist die Hardware-Produktion immer noch das Herz der Computer-Industrie, bei der Suche nach einer interessanten Beschäftigung gewinnen die Spezialgebiete aber zunehmend an Wichtigkeit. Marketing und Vertrieb haben große Zuwachsraten, was die Beschäftigtenzahl angeht. Die Zeichnung stellt die Vernetzung zwischen Herstellern, Händlern, Drittanbietern und dem Endverbraucher dar.

Wartungsingenieure

Die Arbeit des Wartungstechnikers hat noch Ähnlichkeit mit handwerklicher Tätigkeit – die meiste Zeit wird tatsächlich noch „im" Computer verbracht, also bei der Fehlersuche. Dazu muß der Servicetechniker Kenntnisse sowohl aus der Elektronik und Elektrik haben als auch die Grundzüge des Programmierens kennen.

Bezahlung und Arbeitsbedingungen sind meist gut – dagegen steht allerdings auch, daß Techniker viel herumreisen müssen und nicht selten 24 Stunden täglich auf Abruf bereitstehen sollen. Der Angestellte in der Produktion hat es in dieser Beziehung besser.

Mit der gesteigerten Zuverlässigkeit moderner Rechner verliert der Wartungstechniker zunehmend seine Arbeitsgrundlage – Selbstdiagnosesysteme u. ä. übernehmen die Fehlersuche. In naher Zukunft ist damit zu rechnen, daß der Wartungsfachmann nur noch Ersatzteile ausliefert und sie dort einsetzt, wo der Computer im Selbsttest einen Fehler angezeigt hat.

Computerspiel-Autoren

Wenn es sie je gegeben hat, dann ist spätestens heute die Zeit vorbei, in der Teenager mit guten Ideen durch einen Renner auf dem Computerspiel-Markt über Nacht reich werden konnten.

Die meisten Spiele werden von einer recht großen Schar von Freiberuflern geschrieben, die sich ihre Position gleich beim Auftauchen der ersten preiswerten und leistungsfähigen Heimcomputer erkämpft haben.

Nur wenige Programmierer arbeiten allein. Die meisten haben Verträge mit kleinen Software-Firmen oder verkaufen ihre Autorenrechte an Unternehmen, die den Vertrieb für sie übernehmen.

Eine Laufbahn als Freiberufler hat ihre Probleme. Um hinreichend zu verdienen, muß täglich gearbeitet werden – ob man nun gerade Lust dazu hat oder nicht. Freiberufliche Programmierer haben nicht selten einen normalen Beruf und schaffen ihre Produkte in der zweiten Schicht – nach Feierabend. Zudem gab es in den letzten Jahren einige Zusammenbrüche gerade bei kleinen Software-Häusern. Bei der Auswahl des Produktes ist äußerste Vorsicht geboten – Programme für einen Rechner, der nur in geringen Stückzahlen abgesetzt wurde, bringen nichts ein. Schließlich ist das Programmieren von Spielen auch keine echte Laufbahn – man kann es nicht im klassischen Sinne lernen.

Charles Babbage

Obwohl die von Charles Babbage konstruierten Rechenmaschinen nie richtig fertiggestellt wurden, sind sie als die unmittelbaren Vorfahren des heutigen programmierbaren Computers anzusehen.

Wenn sich diese Rechnungen doch um Himmelswillen mit der Dampfmaschine erledigen ließen!" soll Charles Babbage gerufen haben, als er sich mit den Tabellen für den Nautischen Almanach abquälte. Das 19. Jahrhundert hatte der Menschheit zwar die Dampfmaschine beschert, aber die exakte Navigation war in der christlichen Seefahrt immer noch ein Problem: Die Schiffsposition wurde aus der Beobachtung von Sonne, Mond und Sternen anhand mathematischer Tafeln berechnet, die oft ziemlich fehlerhaft waren.

Im Jahr 1812 kam Babbage die Idee, eine Maschine für die umständliche Berechnung der nautischen Tafeln zu bauen. 1823 hatte er ein kleines Modell seiner „Difference Engine" fertig und bat die Regierung um einen Zuschuß für die Finanzierung einer betriebsfähigen Anlage. Der Schatzkanzler gab ihm 1500 Pfund, und Babbage machte sich an den Bau des Rechners.

Dieses Projekt bestimmte Babbages weiteren Lebensweg. Es verschlang Unsummen, denn rein technisch lag es absolut an der Grenze des damals Machbaren. Die erforderlichen Gelder kamen durch Unterstützung des Premierministers, des Duke of Wellington, zusammen. Aber trotz Babbages fester Zusicherung, die Maschine würde ‚alles, was sie tue, garantiert fehlerfrei erledigen' zog sich die Regierung zurück, nachdem sie 17 000 Pfund riskiert hatte. Auch Babbages Mechaniker verschwand bald darauf nach einer heftigen Auseinandersetzung und nahm sämtliche Werkzeuge mit, die speziell für den Bau der Maschine angefertigt worden waren.

Statt zu resignieren, wandte sich Babbage kurzerhand einem noch ehrgeizigeren Projekt zu, nämlich der „Analytical Engine", die noch weit mehr als die „Difference Engine" leisten sollte. Das Konzept wies bereits entscheidende Merkmale des modernen Computers auf: Es waren ein großer Zahlenspeicher und eine Rechen-‚Mühle' (Mill) vorgesehen, die etwa der CPU entsprach, ferner ein Druckwerk für die Ausgabe und vor allem die Möglichkeit der Programmierung unter Verwendung bedingter Verzweigungen.

Für die Befehlssteuerung waren zunächst Stachelwalzen wie bei der Drehorgel gedacht. Später wurde das Lochkartensystem übernommen, das Joseph Jacquard für Webstühle eingeführt hatte. Babbage experimentierte auch mit unterschiedlichen Basiswerten für das Zahlensystem der Maschine, aber da er ganz auf Mechanik fixiert war, konnte er im Binärsystem (schon 1703 von Leibniz vorgeschlagen) keinen wesentlichen Vorteil sehen.

Zusammenarbeit mit Ada

Babbage arbeitete bei diesem Projekt mit der Gräfin Ada Lovelace zusammen, einer begabten Mathematikerin. Beide hatten mit ungeheuren Schwierigkeiten zu kämpfen, nicht zuletzt mit Geldmangel. Die Gräfin verlor auch noch einen großen Teil ihres Besitzes durch ein „unfehlbares" Wettsystem für Pferderennen. Nachdem sie im Alter von 36 Jahren verstarb, setzte Babbage das Werk allein fort.

Als Mann von erstaunlicher Energie erfand er nebenher das Ophtalmoskop für die ärztliche Augenspiegelung, eine Technik für die Signalübermittlung auf See und ein Bühnen-Beleuchtungssystem; außerdem besorgte er die Choreografie für ein Ballett. In seinen letzten Lebensjahren wurde er eigensinnig: Da er den Adelstitel ‚Peer' erwartet hatte, lehnte er die Würde eines „Baron" ab, die man ihm in Anerkennung seiner Verdienste anbot.

Babbage nahm mit seinen Arbeiten das Konzept der heutigen programmgesteuerten Rechenmaschine vorweg, ohne sich der Tragweite seiner Ideen bewußt zu sein.

1792 Geboren in Totnes/Devon am 26. Dezember
1810 Geht ans Trinity College in Cambridge, um dort Mathematik zu studieren
1814 Heiratet Georgina Whitemore
1822 Veröffentlicht einen Artikel mit dem Titel ‚Überlegungen zum Einsatz von Maschinen für die Berechnung mathematischer Tafeln' und erhält die erste Goldmedaille der Astronomischen Gesellschaft, deren Mitbegründer er war
1827 Cambridge beruft ihn auf den gleichen Lehrstuhl, den einst Newton innehatte, bei 80 Pfund Gehalt jährlich. Babbage läßt sich in Cambridge jedoch nicht nieder und hält auch nie Vorlesungen
1833 Parlamentskandidat in Finsbury
1834 Aufgabe der ‚Difference Engine', nachdem sich der Mechaniker Joseph Clement zurückgezogen hat
1862 Ausstellung der teilweise fertiggestellten ‚Difference Engine' in South Kensington/London
1871 Gestorben am 18. Oktober

Duell mit Karten

Herman Hollerith und James Powers leisteten entscheidende Beiträge zur Entwicklung der Lochkartentechnik.

James Powers Maschinen arbeiteten rein mechanisch und waren starr für eine bestimmte Anwendung konstruiert.

Herman Hollerith erfand den elektromechanischen Kartenleser, aus dem später die Tabelliermaschinen entwickelt wurden.

Aus den Lochkartenmaschinen, die Herman Hollerith für die Auswertung der amerikanischen Volkszählung von 1890 konstruiert hatte, entstanden bald die „Tabelliermaschinen" als universelle Datenverarbeitungsgeräte. Bis zur Einführung der ersten Elektronenrechner in den fünfziger Jahren spielten die Tabelliermaschinen eine große Rolle für die zunehmende Rationalisierung in Handel und Industrie. In Pittsburgh/Pennsylvania (USA) experimentierte schon in den dreißiger Jahren ein führendes Kaufhaus mit einem Kundenkonten-System, bei dem 250 auf den Etagen verteilte Terminals über Telefonleitungen mit einer Tabellierer-Zentrale in Verbindung standen. Zur Warenauszeichnung dienten Lochetiketten, deren Daten an die Zentrale fernübertragen wurden, wo der Rechnungsbeleg erstellt wurde und über einen Fernschreiber die Freigabe an das Terminal erfolgte.

Den eigentlichen Anstoß zur Weiterentwicklung der Lochkartenmaschinen gab schon zwei Jahrzehnte vorher die Konkurrenzsituation: Mit Holleriths Monopol für die teuren Volkszählungsgeräte war es 1910 vorbei, als die zuständige Behörde James Powers als Zweitanbieter hinzuzog. Powers offerierte ein rein mechanisches System, das Holleriths Elektromechanik-Patente nicht verletzte. Der folgende Wettstreit zwischen beiden und ihren Firmen stellte für die Entwicklung der Datenverarbeitungstechnik einen Ansporn dar.

1902 entwarf Hollerith ein Steckbrett, mit dem man vorwählen konnte, welche Spalten der Lochkarten zusammengefaßt und als Summe ausgedruckt werden sollten. Dadurch boten Holleriths Maschinen eine gewisse Programmierbarkeit, die der Konkurrenz abging – die Geräte von Powers waren für eine bestimmte Anwendung konstruiert. Im Gegenzug ließ sich Powers 1924 ein System zur Darstellung alphanumerischer Information auf Lochkarten patentieren, das in jeder Spalte genau ein Loch zur Angabe einer Ziffer und eine Lochkombination für jeden Buchstaben verwendete. Hollerith reagierte darauf umgehend mit seinem 80-Spalten-Code, der heute noch Standard ist. Jede Spalte einer Karte enthält zwölf Stanzpositionen, die damals mit „Bürsten" aus Draht elektrisch abgetastet wurden: Wo die Karte gelocht war, bekam die Bürste Kontakt mit einer Metallplatte.

Die ersten Tabelliermaschinen konnten nur zählen oder addieren und beherrschten noch keine mathematischen Funktionen für die anspruchsvollere Verarbeitung. Das änderte sich bald infolge der technischen Fortschritte, und später wurden spezielle Tabelliermaschinen für die Analyse von Schwingungsvorgängen und astronomische Berechnungen eingesetzt.

Lochkartenmaschinen

Auf dem Höhepunkt der Entwicklung zu Beginn der fünfziger Jahre gehörten zu einer Lochkartenanlage mindestens acht verschiedene Stationen. Das Ablochen der Daten erfolgte auf einem Kartenlocher mit Geschwindigkeiten bis zu 200 Karten pro Stunde. Für die Kontrolle der damaligen Datentypisten war eigens ein Prüflocher vorgesehen. Der Lochschriftübersetzer bedruckte die Karten am oberen Rand in Klarschrift, um die Handhabung zu erleichtern.

Die eigentliche Tabelliermaschine summierte spaltenweise die Informationen auf den Lochkarten (bis zu 9000 pro Stunde) und druckte die Ergebnisse in Tabellenform aus – daher der Name. Häufig war auch noch ein Rechenstanzer angeschlossen, der für kompliziertere Arithmetik wie Multiplikationen zuständig war. Der Mischer konnte zwei Lochkartenstapel inhaltlich vergleichen und auch mischen, und der Sortierer machte aus einem Lochkartenstapel bis zu 13 getrennte Haufen – einen für jedes der zwölf Löcher einer bestimmten Spalte und noch einen für ungelocht.

Die Programmierung der Tabelliermaschinen erfolgte über Steuerzeichen auf den Stanzpositionen 11 und 12, wobei die Steuerkarten zum besseren Auffinden im Stapel leuchtend bunt gefärbt waren. Stieß die Tabelliermaschine auf so eine Karte, bedeutete das eine neue Verarbeitungsvorschrift.

Mama Bell

Die Bell-Laboratories liefern seit Jahrzehnten entscheidende Entwicklungsbeiträge für die Hard- und Software. Und sie schrieben ein Stück Computergeschichte.

Schon vor hundert Jahren amüsierte sich Queen Victoria über eine neue Erfindung, dank der sie von der Isle of Wight im Ärmelkanal aus mit ihren Ministern im fernen London sprechen konnte. Vom ersten Telefongespräch der Königin bis hin zur weltweiten Satellitenkommunikation bedurfte es aber noch erheblicher Anstrengungen, wobei der Computer quasi als Abfallprodukt entstand. Zur Förderung der Fernsprechtechnik hatte die „American Telephone & Telegraph Company" (AT&T), eine Gründung Graham Bells, bereits 1925 in Murray Hill/New Jersey (USA) die „Bell Laboratories" ins Leben gerufen, die später unter dem Namen „Mama Bell" bekannt wurden.

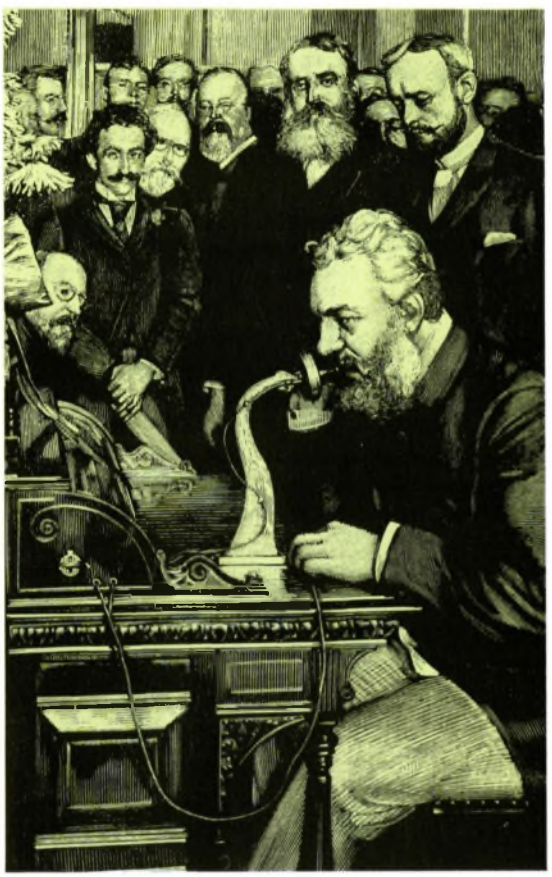
Der Schwerpunkt dieser Einrichtung mit über zwanzigtausend Mitarbeitern liegt in der Grundlagenforschung. Die Wissenschaftler des Labors werden von den Problemen der Tagesentwicklung mit Bedacht abschirmt. Hervorragende Fachleute können tun und lassen, was Ihnen für ihre Forschung wichtig erscheint.

Nobelpreise

Kommen dann dabei innovative Entdeckungen heraus, hat sich der Einsatz des Geldes für das Unternehmen gelohnt. Diese Philosophie brachte für die forschenden Wissenschaftler tatsächlich mehrere Nobelpreise und zwanzigtausend Patente auf den verschiedensten Gebieten ein. Für die Entwicklung des Computers waren viele Arbeiten der Forscher dieses Unternehmens von großer Bedeutung.

In den dreißiger Jahren wurde zunehmend die Fernsprechtechnik weiterentwickelt; vor allem wurde die Vermittlungs-Automatisierung vorangetrieben. Der Verbindungsaufbau erfolgte bereits mit Hilfe eines digitalen Wählcodes, der als eine Folge von Rechteckimpulsen beim Ablauf der Wählscheibe erzeugt und an die Zentrale gesendet wurde. Die Information übernahm dort zunächst ein Relaisregister, bis die Verbindung über ein Kreuzschienensystem durchgeschaltet war; die Wählimpulse mußten gezählt und in Koordinaten auf elektromechanischen Schaltfeldern umgesetzt werden. Mit diesem Vermittlungssystem standen die wesentlichen Rechnerkomponenten eigentlich schon bereit – es mußte nur noch der rechte Mann kommen.

Dieser Mann war der Bell-Mathematiker Georg Stibitz, dem die Analogie zwischen dem



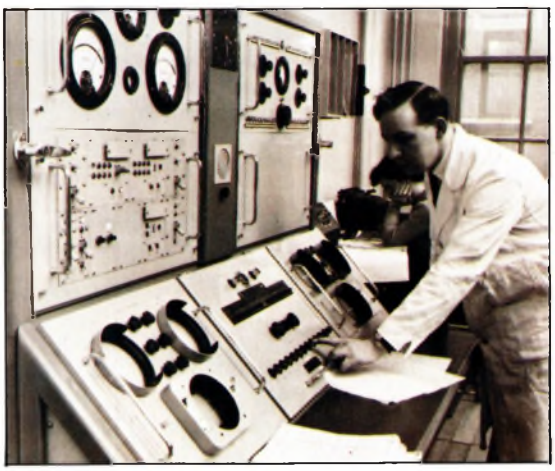
Die Bell Laboratories sind nach dem Sprachheillehrer Alexander Graham Bell benannt, der als schottischer Auswanderer in Amerika 1876 das erste brauchbare Telefon erfand. Das Prinzip der elektrischen Tonübertragung hatte schon 1861 der Deutsche Philipp Reis vorgeführt. Nach der Überlieferung bestand das erste Telefongespräch Bell's aus dem nachfolgenden bedeutenden Satz, den er an den Assistenten im Keller gerichtet hatte: „Kommen Sie mal rauf, Mr. Watson, ich brauche Sie hier!"

„Zählen" der Wählimpulse und dem Vorgehen beim Addieren auffiel. Auf seinem Küchentisch bastelte er aus einem alten Koordinatenwähler und ein paar Relais die ersten elektromechanischen Computer-Funktionsgruppen.

Stibitz tat sich dann mit dem erfahrenen Vermittlungs-Ingenieur Samuel B. Williams zusammen, der schon seit Jahrzehnten die Kreuzpunktschalter im Kopf hatte, und baute mit ihm den „Complex Number Calculator". Dieses Gerät war eine Rechenmaschine für „komplexe" Zahlen, die ja bekanntlich imaginäre Anteile enthalten und bei der Lösung von Polynomgleichungen gebraucht werden. Der Bau des Apparates begann 1937, und man benötige 450 Relais sowie zehn Kreuzschienenfelder, bis er am 8. Januar 1940 erstmals arbeiten konnte. Er arbeitete mit binärer Codierung und benötigte für die Division bei zwei achtstelligen Zahlen eine halbe Minute. Im September 1940 wurde diese Rechengeschwindigkeit der Fachwelt vorgeführt.

Tee oder Computer

Der kommerzielle Einsatz des Computers begann auf der britischen Insel ausgerechnet in einem Teegeschäft.



Anders als seine Vorläufer, die für wissenschaftliche oder militärische Zwecke konstruiert wurden, war der LEO 1 nur für einfachste Arithmetik gedacht, konnte aber viele tausend Geschäftsvorgänge pro Tag bearbeiten.

An einem ungewöhnlichen Ort, nämlich in einer englischen Teehandlung, fiel 1947 die bahnbrechende Entscheidung, einen Rechner für die Automatisierung von Büroarbeiten zu entwickeln. Diesen Entschluß wagte das Unternehmen ‚J. Lyons Ltd.', dessen Ladenkette internationalen Ruf genoß. Kennzeichnend für ein derartiges Geschäft sind zahllose kleine Vorgänge, und um die Abwicklung überhaupt rentabel zu gestalten, muß der zugehörige Papierkram möglichst effizient bearbeitet werden.

Die Firma hatte schon eine gewisse Tradition in der Innovation der Bürotechnik: Bereits 1896 bekamen die Läden Rechenmaschinen, und in den dreißiger Jahren experimentierte Lyons mit Mikrofilm-Dokumentationsverfahren. Damals wurde auch erstmals ein eigenes Büro-Forschungszentrum zur Untersuchung von Arbeitsmethoden gegründet.

Das Unternehmen schickte des öfteren Mitarbeiter aus, die sich nach interessanten neuen Entwicklungen umsahen. 1947 kamen zwei Lyons-Mitarbeiter in die USA, um in den geheimnisvollen „Elektronengehirnen" herumzuschnüffeln. Sie kehrten mit der Erkenntnis zurück, daß sie sich die Reise hätten sparen können, da auch vor ihrer Haustür, an der Universität Cambridge, Computer entstanden.

Der Lyons-Vorstand ließ daraufhin durch eine Untersuchung klären, ob sich die Entwicklung eines eigenen Rechners für die Firma lohnen würde – den Entwicklungskosten von £ 100 000 stand eine jährliche Einsparung von £ 50 000 gegenüber. Folgerichtig gab Lyons im Oktober 1947 den Startschuß.

Die traditionsreichen Teehandlungen der englischen Firma Lyons & Co. gaben äußerlich eigentlich nicht das passende Milieu für den ersten kommerziellen Computereinsatz ab. Genau dieses Vertriebsgeschäft mit sehr vielen kleinen Umsätzen war aber prädestiniert für die Einführung der elektronischen Datenverarbeitung.

Primzahlen-Tabelle

1949 war es dann soweit, der Computer bewältigte eine akademische Aufgabe: die Erstellung einer Primzahlen-Tabelle.

Lyons analysierte derweil die Aufgaben, für die der neue Rechner gedacht war, und skizzierte die erforderlichen Programmabläufe. Diese Studien lieferten die Basis für spätere Software und für die Hardware: Es zeigte sich bald, daß ein Computer für den Büroeinsatz anders aussehen mußte als einer für die Wissenschaft. Während der Computer nach Eingabe von wenigen Zahlen einen komplizierten Formelsalat langwierig numerisch auszuwerten hatte, sollte der Bürorechner praktisch das Gegenteil tun – bei minimalem mathematischem Aufwand, beschränkt auf die Grundrechenarten, waren viele Daten zu bewältigen.

Am 9. Februar 1954 trat LEO (Lyons Electronic Office) erstmals in Aktion und erledigte die wöchentliche Lohnabrechnung für die 1700 Lyons-Mitarbeiter. Einen Vorgang, für den ein Angestellter zuvor acht Minuten gebraucht hatte, schaffte LEO in eineinhalb Sekunden.

LEO erwies sich als großer Erfolg, so daß nicht nur Lyons, sondern auch andere Unternehmen bald an diesem Gerät interessiert waren. Lyons gründete eine Tochtergesellschaft namens ‚LEO Computers', um das vorhandene Know-how auszuschlachten. Sie erzielte hervorragende Geschäftsergebnisse, indem sie eine Reihe verbesserter LEO-Versionen auf den Markt brachte.

Fachwörter auf einen Blick

ASCII
American Standard Code for Information Interchange – Standardisierter Code zur Darstellung von Zeichen im Computersystem

Baud-Rate
Geschwindigkeit bei der Datenübertragung

Binärsystem
Rechensystem, das auf den Ziffern 0 und 1 aufgebaut ist

Bit
Binary Digit – Binärziffer (1 oder 0)

BTX
Bildschirmtext, Postmedium für Informationen

Byte
Gruppe von acht Binärziffern (Bits)

CAD
Computer Aided Design – Computer-unterstütztes Entwerfen

Chip
Siliziumplättchen mit integriertem Schaltkreis

CP/M
Control Programm for Microcomputers – Betriebssystem, das aus drei Programm-Moduln besteht

CPU
Central Processing Unit – Zentraleinheit, bildet das Verarbeitungszentrum des Computers

Datenbank
Spezielle Anordnungen von Daten auf Massenspeichern

Floppy-Disk
Kunststoffscheibe mit magnetisierbarer Oberfläche zum Speichern von Daten

Hardware
Teile des Computers, die man sehen und anfassen kann

Hauptspeicher
Interner Speicher aus Speicher-Chips zusammengestellt

Interface
Schnittstelle, Verbindungsstelle zwischen zwei Bauteilen

Joystick
Steuerknüppel für Computerspiele

Konsole
Zur Bedienung eines Computers notwendige Zusatzgeräte wie Tastatur und Bildschirm (Ein- und Ausgabeeinheiten); bei Microcomputern auch Frontplatte

Laserdrucker
Bis zu zehnmal schneller als herkömmliche Drucker; die Zeichen werden per Laserstrahl produziert

Laufwerk (drive)
Antriebseinheit für Speicher. Bezeichnet auch die gesamte Speichereinheit; im Laufwerk befinden sich Speichermedien wie Festplatte, Diskette, Magnetband usw.

Magnetband
Eignet sich gut zum Speichern von Daten; in der Computertechnik werden Magnetbänder oft zur Datensicherung verwendet

KByte
KiloByte – genau 1024 Bytes

Maus
Eingabegerät, das die Tastatur oder den Joystick ersetzt

Micro-Drive
Massenspeicher der Firma Sinclair, Mittelding zwischen Cassette und Floppy

Microprozessor
Zentraleinheit auf einem oder zwei Chips

Modem
Gerät, das binäre Informationen in Tonsignale umsetzt; somit können Informationen per Telefon ausgetauscht werden

Peripherie
Zusatzgeräte im Computersystem mit spezifischen Aufgaben, z. B. Joystick

Plotter
Computergesteuertes Zeichengerät für Strichzeichnungen

RAM
Random Access Memory – Schreib/Lesespeicher, aus dem Daten gelesen und eingegeben werden können

ROM
Read Only Memory – Festwertspeicher, dessen Daten nicht verändert werden können

Software
Programme, die zur Benutzung des Computers notwendig sind

Steckmodul
Zusammenfassung mehrerer Bauelemente zu einer Funktionseinheit

Track Ball Controller
Steuergerät für Spiele, bei dem die Daten mit einer Kugel eingegeben werden

Fachwörter auf einen Blick

Adapter
Ein Zwischenstecker, der unterschiedliche Stecker verbindet

Analog-Digital Wandler
A/D-Wandler – eine Schaltung, bei der ein analoges Eingangssignal in eine digitale Darstellung umgewandelt wird

Auflösung
Die Anzahl der Punkte, die auf einem Bildschirm zur Verfügung stehen

Bildwiederholspeicher
Auf dem Monitor muß das Bild immer wieder aufgebaut werden. Der Bildwiederholspeicher hält die Bildinformationen bereit

Cursor
Leuchtzeichen auf dem Bildschirm, das angibt, an welcher Stelle auf dem Bildschirm weitergeschrieben werden kann

Display
Anzeige für den Computer. Häufig verwendet wird das LCD-(Flüssigkristall) Display

I/O
Input/Output – Ein- oder Ausgaben zur Peripherie

Interrupts
Der Microprozessor wird für andere Aufgaben unterbrochen

Keyboard
Englischer Ausdruck für Computertastaturen

Lightpen
Optischer Lesestift, mit dem Informationen direkt in den Rechner eingegeben werden

Magnetplatte
Schneller Massenspeicher mit sehr hoher Kapazität; auch häufig als pauschaler Begriff für Platten und Floppy

Matrix
Struktur der Anschlußdrähte für die Tasten des Keyboards

Megabyte
Ein Megabyte entspricht dem Wert von 1024^2 Bytes

Menü
Am Bildschirm erscheinende Tabelle von Computerfunktionen, die dem Anwender zur Auswahl stehen; durch Eintippen der Ziffer, die dem gewünschten Programm zugeordnet ist, läßt sich dieses über die Tastatur aufrufen

Micro-Floppy
Kunststoffscheibe (Diskette) mit sehr großer Speicherkapazität und unterschiedlichem Durchmesser

Multiskating
Mehrere voneinander unabhängige Programme laufen zur gleichen Zeit auf einem Computer

Paddles
Drehknöpfe für Computerspiele; nicht ganz so bedienungsfreundlich wie Joysticks

PASCAL
Programmiersprache, benannt nach Blaise Pascal (1642)

Potentiometer
Veränderbarer elektr. Widerstand

QWERTZ
Die erste Buchstabenfolge einer Schreibmaschinentastatur, daher QWERTZ-Tastatur

RESET-Taste
Sie löscht den gesamten Speicherinhalt im RAM

Schaltung, Integrierte
Eine vollständige Schaltung ist auf einem Chip untergebracht

Scrolling
Der Bildschirm wird „gerollt", der Inhalt wandert entweder nach oben oder unten, links oder rechts

Software-Paket
Eine Zusammenstellung verschiedener Programme, die oft günstiger angeboten werden

Terminal
Datenstation – Peripheriegerät, das mindestens aus Bildschirm und Tastatur besteht und oft eine Telefonleitung zur Kommunikation mit dem Hauptrechner benutzt

Transistor
Halbleiterbauelement als elektronischer Schalter

Übertragungsgeschwindigkeit
transfer rate – während einer bestimmten Zeit übertragene Bit-Zahl (Baud-Rate)

Unterprogramm
subroutine – Programm, das zur Ausführung von sich wiederholenden Funktionsabläufen vom Hauptprogramm aufgerufen wird

Word Star
Bekanntes Programmsystem für die Textverarbeitung

Zeichensatz
character set – sämtliche Zeichen, die auf einem Bildschirm oder Drucker ausgegeben werden können

Fachwörter auf einen Blick

BUS
Leitungssystem (Sammelschiene) zur Übertragung von Informationen zwischen Prozessor und mehreren Peripheriegeräten; Busse werden bei Computern in Adreß-, Daten- und Steuer-Busse unterteilt

Centronics
Parallele Schnittstelle für Drucker, die sich als Industriestandard durchgesetzt hat; so sind Drucker mit verschiedenen Computern kompatibel

Compiler
(Kompilierer) Programm, das andere Programme, die in einer höheren Sprache geschrieben sind, in Maschinencode umsetzt

Elektroden
dienen dem Aufbau des Elektronenstrahls in einer Bildröhre; die mit dem positiven Pol der Hochspannungsquelle verbundene Elektrode heißt Anode, die mit dem negativen Pol verbundene heißt Kathode

Gatter
(gate) Logische Schaltungsfunktion wie zum Beispiel die AND-, OR-, NOT-Schaltung

Höhere Programmiersprache
Für spezielle Anwendungsbereiche entwickelte Programmiersprachen wie PASCAL oder COBOL; letztere wird beispielsweise für den kaufmännischen Bereich genutzt; die Sprache FORTRAN eignet sich vor allem für die Bearbeitung von wissenschaftlich-technischen Aufgaben

Interrupt
(Unterbrechung) Der Programmablauf wird unterbrochen, so daß ein anderer Teil des Programms bearbeitet werden kann

Koaxialkabel
Eine elektrische Doppelleitung, wobei das innere Kabel von einem Aluminium- oder Kupferdrahtgeflecht (Außenleiter) umgeben ist; Koaxialkabel eignen sich zum Übertragen breiter Frequenzbänder

kompatibel
Computer sind kompatibel, wenn die Programme auf verschiedenen Rechnern ohne Änderungen laufen

Künstliche Intelligenz
Wenn Computer Tätigkeiten ausführen, die menschliche Intelligenz erfordern (Beispiel Expertensysteme), spricht man von künstlicher Intelligenz

Kurvenschreiber
Auch als Plotter bekannt; computergesteuertes Gerät, das Grafiken auf Papier darstellt

Matrixdrucker
Die zu druckenden Zeichen werden aus einzelnen Punkten zusammengesetzt (Punktmatrix)

Operator
Symbole oder Buchstaben, welche die durchzuführenden arithmetischen und logischen Operationen angeben, wie + für Addition oder die Buchstaben OR für die Oder-Funktion

David Higham

Paralleldrucker
Drucker, der komplette Zeilen auf einmal ausdruckt

RS232
Standard-Schnittstelle (in den USA genormt) für seriellen Datentransfer zwischen Peripherien; das europäische Gegenstück ist die V24-Schnittstelle

Speech-Editor-Modul
Sprachausgabe-Modul – ermöglicht das Umsetzen von Buchstaben in Sprache, die über Lautsprecher ausgegeben wird

Thermodrucker
Die Abbildung der Druckzeichen erfolgt auf besonderem Thermopapier durch Erwärmung. Thermodrucker arbeiten sehr leise; bis zu 120 Zeichen können pro Sekunde ausgegeben werden

Tintenstrahldrucker
(anschlagsfreier Drucker) arbeiten geräuscharm und schnell mit guter Druckqualität; durch feine Düsen werden Tintentropfen (mit 0,1 mm Durchmesser) auf das Papier gespritzt, die einen 0,3 mm kleinen Fleck hinterlassen; sie gehören zu der Familie der Matrixdrucker

Tuner
Abstimmvorrichtung für die Einstellung einer bestimmten Empfangsfrequenz in der Hochfrequenztechnik; in Fernsehgeräten dient er zur Kanalauswahl

Typenraddrucker
(Ganzzeichendrucker) Um einen Zylinder herum befinden sich sternförmige Typenarme, an denen die einzelnen Zeichen eingeprägt sind; gute Schriftqualität, allerdings langsamer als Matrixdrucker und ein hoher Geräuschpegel

Fachwörter auf einen Blick

Absturz
(system crash) Als Absturz bezeichnet man den Ausfall des Systems; dies kann die Folge sowohl von Fehlern bei der Hardware als auch der Software sein

arithmetischer Befehl
Befehl für arithmetische Operationen, zum Beispiel Addition und Multiplikation

Code
Informationen, die in den Rechner eingegeben werden, sind in Computercode gespeichert; Beispiele: ASCII, EBCDIC und BCD

Disk Operating System
(DOS) Oft verwendetes Betriebssystem (Software), das auch die Diskettenlaufwerke verwalten kann

Dualzahl
Zahl, die auf einer Potenz von 2 aufgebaut ist

EDV
Elektronische Datenverarbeitung

Endlospapier
(fan-fold paper) Papier für Drucker, das mit Randlochung versehen ist; die einzelnen Seiten sind durch Perforation miteinander verbunden

Einchip-Microcomputer
Auf einem Chip integriert befinden sich Zentraleinheit, Programm- und Datenspeicher, Taktgeber, Zähler sowie Ein- und Ausgabeeinheiten

Handheld-Computer
Kleine Computer, die in Aktentaschen passen; diese handlichen Rechner können mit Batterien betrieben werden; zahlreiche Peripheriegeräte, wie zum Beispiel Diskettenlaufwerk und Bildschirm können angeschlossen werden

Kathodenstrahlröhre
Durch einen Elektronenstrahl wird auf der Stirnseite der Kathodenstrahlröhre ein Bild gezeichnet, das aus einzelnen Rasterpunkten aufgebaut wird

Magnetblasenspeicher
(Bubble-Speicher) Datenträger von extrem hoher Speicherkapazität im Format eines Chips; die im Magnetblasenspeicher aufbewahrten Daten bleiben auch nach dem Ausschalten des Gerätes erhalten; die Datenübertragungsgeschwindigkeit ist höher als von Diskettenlaufwerken

Microdrive
Spezieller Datenrecorder, der über ein Interface an den Sinclair-Heimcomputer angeschlossen werden kann

MicroSoft
Die Firma MicroSoft bietet Programme (zum Beispiel MicroSoft-BASIC) an, die auf vielen Personal-Computern laufen

Minidrucker (Microwriter)
werden oft in Handheldcomputern verwendet; sie arbeiten sehr geräuscharm und schnell auf thermischer Basis; für den Druck wird Spezialpapier benötigt

Pixels
(picture cell) Bildpunkte auf dem Monitor

Portable
Computer, die wie ein Aktenkoffer getragen werden können, werden als Portables bezeichnet

Schreib/Lesekopf
(Read/Write Head) Der Schreib/Lesekopf eines Diskettenlaufwerks löscht, liest und schreibt Daten auf Diskette; diese dreht sich mit hoher Geschwindigkeit unter dem Schreib/Lesekopf

Strichcode
(bar code) Vor allem bei Konsumgütern verwendeter Code, der aus Linien unterschiedlicher Stärke besteht; mit einem Lesestift werden die darin enthaltenen Informationen erkannt und ausgewertet

SuperCalc
Software für Tabellenerstellung, Kalkulation und Finanzplanung

Takt
(clock) Computer werden taktgesteuert betrieben; die Impulse liefert ein Taktgenerator, um die Synchronisierung aller Vorgänge zu gewährleisten

TTL
(Transistor Transistor Logic) In diesen Schaltkreisen werden die Transistoren direkt gekoppelt

Twin Floppy Plattenbetrieb
Zwei Disketten-Laufwerke in einem Gehäuse teilen sich die Arbeit: Auf der einen sind z. B. die Programme, auf der anderen die Daten

Vocoder
Ein System zur Erzeugung von künstlicher menschlicher Sprache; es wird zum Beispiel bei der Fahrplanauskunft eingesetzt

Fachwörter auf einen Blick

A/D Wandler
Analog-Digital-Wandler; Gerät, das eine analoge Eingabe in digitale Werte umwandelt

Akustikkoppler
ermöglichen die Datenübertragung über weite Entfernungen. Akustikkoppler sind passende Telefon-Sprechmuscheln, die an den Computer angeschlossen werden können

Directory
Verzeichnis, das angibt, an welcher Stelle sich abgelegte Programme oder Daten auf Diskette oder Cassette befinden

Duplex
Übertragungsart, bei der Daten gleichzeitig in beide Richtungen übertragen werden können; hierzu wird ein Übertragungssystem mit zwei Leitern verwendet

Elektron
Leicht negativ geladenes, stabiles Elementarteilchen

Flußdiagramm
Grafische Darstellung eines Programmablaufs; wird zur Erstellung von Rechnerprogrammen benutzt

FTZ-Nummer
FTZ ist die Abkürzung für „Fernmelde-Technische Zulassung", die für alle Geräte erforderlich ist, die in die Posthoheit eingreifen; ein Modem oder ein Akustikkoppler bedient sich des Telefonnetzes und muß daher von der Post mit einer FTZ-Nummer zugelassen sein

Halbleiter
Kristalline Stoffe, die sich bei niedrigen Temperaturen wie Isolatoren verhalten; ihre Leitfähigkeit nimmt bei steigender Wärme zu

LED
(Light Emitting Diode) Leuchtdiode, die als Anzeigeelement dient; LEDs existieren in mehreren Farben

Mailbox
Elektronischer „Briefkasten" in einem größeren Computer, der über Telefon oder Datex-Leitungen mit Hilfe eines Modems angewählt werden kann; in der Mailbox können Nachrichten für andere abgelegt oder aber Informationen abgefragt werden

Microcassetten
Für Diktiergeräte und Handheld-Computer verwendete Mini-Cassetten

Platine
Leiterplatte, auf der verschiedene Bausteine befestigt werden; besteht in der Regel aus Kunststoff

Raster
Senkrechte und waagerechte Einteilung des Bildschirms; Monitorbilder sind gerastert

Silizium
Chemisches Element, das für die Halbleiterelektronik sehr wichtig ist; es wird in einem aufwendigen Fertigungsprozeß aus Quarzsand gewonnen

Sonderzeichen
Zeichen und Symbole, die zusätzlich im Zeichensatz vorhanden sind

Start-Bit
In der Datenfernübertragung werden die seriell übertragenen Bytes mit einem Start- sowie mit einem Stopbit versehen, damit der empfangende Rechner erkennen kann, wann ein Byte beginnt und wann es endet; zusätzlich werden oft noch Prüfbits angehängt, um die Übertragung zu überprüfen

Telefax
Telekopierer, der über die normale Telefonleitung Texte und Grafiken übermitteln kann; benötigt wird dazu ein druckerähnliches Gerät

David Higham

Überschreiben
Eingabe von Daten in den Speicher, wobei die bereits abgespeicherten Daten gelöscht und zerstört werden

User
Anwender eines Rechners werden in der Computersprache User genannt (englisch „to use")

Vektor
Höherstehende Zahl, die neben dem Zahlenwert (Größe) auch durch eine Richtung gekennzeichnet ist

Vollduplex
Art der Datenübertragung, bei der Leiter für alle Übertragungsrichtungen vorhanden sind

Wafer
Siliziumscheibe, auf der integrierte Schaltkreise aufgetragen sind; wenn integrierte Schaltkreise montiert sind, werden die einzelnen Plättchen in ein Gehäuse eingebaut; der Microprozessor eines Computers ist ein Wafer

Fachwörter auf einen Blick

Adreßleitungen
Sind über Adreßkontakte mit der CPU verbunden und dienen zum Transport von codierten Informationen

Ausführungszeit
Das bestimmte Tempo, mit dem Programme oder Befehle im Computer ablaufen

Benchmark
Bewertungsprogramm zum Testen der Leistungsfähigkeit verschiedener Prozessoren, das die am häufigsten benutzten Befehle beinhaltet und einen Vergleich von Verarbeitungsgeschwindigkeit und Speicherplatzbedarf zwischen verschiedenen Systemen ermöglicht

Buffer
Zwischenspeicher bei Hard- und Software, in dem Daten bis zur weiteren Bearbeitung abgelegt werden können

CAD
(Computer Aided Design) Entwicklung und Entwurfsarbeiten mit Unterstützung des Computers; dabei wird die grafische Darstellung vom Computer übernommen

CGI
(Computer Generated Imaging) Computergesteuerte Wiedergabetechnik von einzelnen Filmsequenzen; die Animation wird häufig zur Produktion von ‚Zeichentrickfilmen' benutzt

Hardcopy
Über Drucker oder auf Microfilm ausgegebene Informationen, die zum Beispiel den Bildschirminhalt duplizieren oder Operationen des Rechners protokollieren

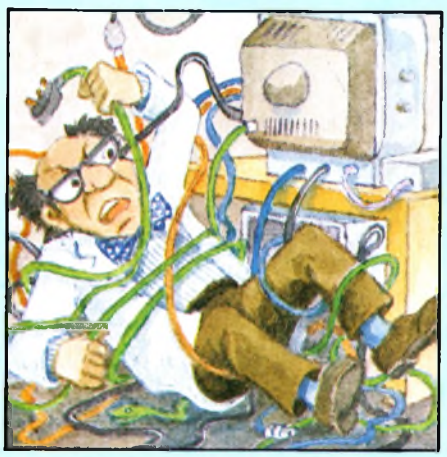
Inverter
Gerät oder Schaltung, womit Gleichstrom in Wechselstrom umgewandelt wird

Invertierung
Umkehrung bzw. negative Darstellung von Buchstaben, Zahlen und Grafiksymbolen

Latch
(Auffang-Flipflop) Spezieller Speicher, der beim Datenaustausch zwischen Peripheriegeräten und Prozessor zur Anpassung der unterschiedlichen Übertragungsgeschwindigkeiten dient

Link Trainer
Flugsimulatoren der ersten Generation zur Schulung von Piloten

parallele Schnittstellen
übertragen die Bits eines Zeichens gleichzeitig auf mehreren Kanälen


David Higham

Paritybit
(Paritätsbit) Prüfbit, das einer Folge von Informationsbits angefügt wird, um Übertragungsfehler feststellen zu können

Personalcomputer
auch Microcomputer oder Heimcomputer genannt; allein die Verfügbarkeit von Programmen und die Speicherkapazität entscheiden über die Einsatzmöglichkeit

Polling
Abfrage, ob die Peripheriegeräte zum Datenaustausch bereit sind

PPI(Programmable Periphal Interface) programmierbare Peripherieschnittstelle

Sequenzer
Bestandteil eines Synthesizers zum Speichern und Abrufen von Tonfolgen

serielle Schnittstellen
übertragen die Bits nacheinander über einen Kanal

single density
und double density, einfache bzw. doppelte Schreibdichte auf Disketten; 'double density' bietet bei entsprechendem Diskettenlaufwerk die größere Kapazität

Slot
Steckanschluß im Computer für Platinen oder Zusatzeinheiten

Softwarehaus
Produzent von Programmen; die angebotene Software wird einzeln oder zusammen mit einem Computer verkauft

Spreadsheet
Tabellenkalkulationsprogramme, die die eingegebenen und berechneten Daten auch grafisch darstellen können

Tortendiagramm
Kreis-Grafik, die zur Veranschaulichung von Daten in Segmente aufgeteilt wird

Treibersoftware
Gerätesteuerungsprogramm im Betriebssystem

Visicalc
(Supercalc/Masterplan), Kalkulationsprogramme, die auf betriebswirtschaftliche Anwendungen ausgerichtet sind; sie erlauben die Weitergabe von Daten der Tabellenkalkulation an Textverarbeitungsprogramme und die Manipulation von Datenbanken

Fachwörter auf einen Blick

CAM
(Computer Aided Manufacturing) – Programme zur Steuerung numerisch gestützter Werkzeugmaschinen

DMA
(Direct Memory Access) – direkter Speicherzugriff auf Daten mittels eines besonderen Steuerelements ohne Einschaltung der CPU

dongle
Modul zum Schutz gegen unbefugtes Kopieren von Programmen

festverdrahtet
Logische Funktionen werden durch festverdrahtete Schaltungen verwirklicht (zum Beispiel Gatter)

Helligkeit
Kann vom Computer-Bediener am Bildschirm verändert werden; beeinflußt die Lesbarkeit der dargestellten Zeichen

Hertz
(Hz) Maßeinheit für die Frequenz; wurde nach dem Physiker Heinrich Hertz benannt

höchstwertiges Bit
Stelle mit dem höchsten Wert eines Bits; zum Beispiel bei binär 1000 ist 1 das Bit mit dem höchsten Wert

Hostrechner
Bezeichnung für eine große Datenverarbeitungsanlage, mit der auch kleinere Computersysteme verbunden werden können

Informatik
Naturwissenschaftlicher Studiengang; Informatik bezeichnet die Lehre der Datenverarbeitung und deren Anwendung

Interpreter
Programm des Betriebssystems zur Abarbeitung von Programmen höherer Programmiersprachen

LAN
(Local Area Network) – Netzwerk von Computersystemen in einem begrenzten lokalen Bereich

Makrobefehl
Unter einem bestimmten Namen wird eine Gruppe von Befehlen aufgerufen; ein sogenannter Makroassemblierer sorgt für die Übersetzung der Namen in Befehle

manuell
mit der Hand betätigt

Multiplex
Datenübertragungsform, wobei verschiedene Sender einen gemeinsamen Übertragungsweg benutzen; anschließend werden die Daten entsprechend ihrer Zugehörigkeit wieder aufgeteilt

Netzwerk
(network) – Zusammenschluß von Rechnern zu einem Datenkommunikationsnetz

Objektcode
Durch Compiler in Maschinencode übersetzte Programme

Optischer Markierungsleser
Peripheriegeräte, die Markierungen (Bleistift, Drucke oder Lochungen) abtasten und diese in Eingabeimpulse umwandeln

Quellcode
(Source code) – in einer Programmiersprache abgefaßtes Programm

Quellprogramm
(Source program) – ursprüngliche Fassung von Programmen, die für den Gebrauch in Maschinensprache compiliert werden müssen

Sonderzeichen
Zeichen und Symbole, die neben Buchstaben und Zahlen im Zeichensatz vorhanden sind

Stopplänge
Bezeichnet die Länge eines Bandes, das am Schreib/Lese-Kopf nach dem abschließenden Schreib- oder Lesevorgang bis zum endgültigen Stopp entlangläuft

Terminal-Interface
Anschlußgerät für Terminals zur Anpassung von Geschwindigkeit und Spannungspegel zum Rechner beim Datentransfer

Transferrate
Übertragungsgeschwindigkeit für Daten

Überschreiben
Eingabe von Daten in den Speicher, wobei die bereits abgespeicherten Daten gelöscht und zerstört werden

verschieben
(shift) – Verschieben binärer Daten in einem Datenregister

Winchesterplatte
Festplatten-Speicher mit hoher Informationsdichte und kurzer Zugriffszeit

Zustandsbit
Bit, das vom Computer auf 0 oder 1 gesetzt wird, um einen bestimmten Zustand anzuzeigen; zum Beispiel um festzuhalten, ob ein Zusatzgerät angeschlossen ist (busy) oder nicht

Fachwörter auf einen Blick

Abruf
(fetch) Vorgang, bei dem Daten aus einem Speicher geholt werden

ADA
Höhere Programmiersprache für Echtzeit-Anwendungen; ADA wurde 1980 in den USA entwickelt

ALGOL
Problemorientierte höhere Programmiersprache für Aufgaben im technisch-wissenschaftlichen Bereich; sie wurde Anfang der 60er Jahre entwickelt; ALGOL ist die Abkürzung für ALGOrithmic Language (algorithmische Sprache)

Algorithmus
Bezeichnung für einen Rechenvorgang, der genau nach einem bestimmten Schema abläuft; algorithmische Computersprachen wie FORTRAN, COBOL und ALGOL beschreiben die Rechenvorgänge, die mit Hilfe von Algorithmen möglich sind

alphanumerisch
Ein alphanumerischer Zeichensatz besteht aus Buchstaben, Ziffern und Sonderzeichen

Anschluß
(port) Anschlußstelle von Peripheriegeräten an das Bus-System des Microprozessors

ANSI-Norm
Technische amerikanische Norm, ähnlich der DIN-Norm in Deutschland; das American National Standards Institute entwickelt und veröffentlicht die ANSI-Normen

CAL
(computer assisted learning) computerunterstütztes Lernen; Einsatz des Computers im Schul- und Lehrbereich, um das Lernen zu rationalisieren, Antworten zu überprüfen und dem Lernenden Hinweise zu geben

COBOL
Abkürzung für Common Business Oriented Language (kaufmännisch orientierte Programmiersprache)

Cartridge
Speichermedium mit seriellem Zugriff

Digit
Ziffer, Zeichen, Stelle

Digitiser
Grafiktablett; es setzt geometrische Positionen in digitalisierte Informationen für den Computer um

DNA (DNS)
Desoxiribonuclein-Acid(-Säure); Träger der Erbinformationen in den Genen von Pflanze, Tier und Mensch

Index-Datei
Dies ist eine spezielle Dateistruktur, bei der den verschiedenen separaten Dateien Indexdateien zugeordnet sind; eine indizierte Datei versetzt den Computer in die Lage, genau festzustellen, an welcher Stelle auf dem Datenträger sich bestimmte Daten befinden, zum schnellen Abruf

Megabyte
1 Megabyte entspricht 1 048 576 ($=2^{20}$) Byte

Modulator
Er wandelt elektrische Signale um, die gesendet werden sollen

RPNL
(Reverse Polish Notation Language) Höhere Programmiersprache, die den Sprachen FORTH und PASCAL ähnelt

Supraleitfähigkeit
1911 von H. Kamerlingh Onnes entdeckte physikalische Erscheinung: Bei starker Abkühlung verschwindet der elektrische Widerstand vieler Metalle vollkommen; die Leitfähigkeit nimmt zu; das Innere des jeweiligen Drahtes ist frei von Magnetfeldern

Verschachtelung
(nesting) Technik beim Programmieren, die ein Programm in Subroutinen (Unterprogramme) unterteilt; von diesen Unterprogrammen werden wiederum Unterprogramme aufgerufen

Verschlüsselung
Um Daten auf Datenträgern speichern zu können, werden sie binär verschlüsselt, so daß sie für den Computer verständlich sind

Volt
ist die physikalische Einheit, die der elektrischen Spannung entspricht

Bildnachweise:
225: Richard Williams Animation LTD;
227: Ian Dobbie, Walt Disney LTD;
228, 229: Kevin Jones;
232: Chris Stevens;
233: Rolf Seiffe;
235: Bob Freeman;
236: Tony Lodge;
239: Simon Lewis;
240, 241: Liz Dixon;
241: Ian McKinnel;
242: Caroline Holden;
246: m. frdl. Gen. Birkhäuser Verlag;
250: Royal Society;
251: Kevin Jones.

Fachwörter auf einen Blick

Abakus
Ursprünglich war dies ein mit Sand bestreutes Brett, auf das Zahlzeichen geschrieben wurden, um auf diese Art und Weise zu rechnen; später wurde daraus ein Rechenbrett, auf dem mit Hilfe frei beweglicher Steine Additionen und Subtraktionen durchgeführt wurden; als Weiterentwicklung entstand der heute noch bekannte Holzrahmen mit Drahtstäben, auf die Perlen aufgefädelt wurden; durch Verschieben der Perlen wird das Zusammenzählen und Abziehen verdeutlicht

ADSR
Kurzform für die schnell aufeinanderfolgenden Veränderungen eines Tones beim „Anschlag": Attack, Decay, Sustain und Release

Akkumulator
akkumulatives Register; dieser Speicher des Microprozessors ist bei fast allen Ein/Ausgaben beteiligt

Coprozessor
Um die Leistungsfähigkeit zu verbessern, haben einige Rechner einen zweiten Prozessor, der der CPU Aufgaben abnimmt

Editor
Hilfsprogramm, das Eingabe, Korrektur, Speicherung und Ausgabe von Programmen unterstützt

EEROM
Electrically Erasable ROM; Festwertspeicher, der durch das Anlegen eines Löschimpulses gelöscht und dann neu „beschrieben" werden kann

Eindimensionales Feld
Daten mit gemeinsamen Eigenschaften werden entweder nur untereinander (Spalte) oder nur nebeneinander (Zeile) geschrieben

ELAN
Um 1975 entwickelte höhere Programmiersprache, die auf Erfahrungen mit PASCAL und ALGOL basiert

Fehlersuchprogramm
Dienstprogramm, das die Fehlersuche unterstützt; es stellt eine Reihe von Funktionen zur Verfügung, wie beispielsweise das Setzen von Programmunterbrechungs-Punkten sowie Kontrolle und Änderung der verwendeten Variablen und Register während des Programmablaufs

FIFO-Speicher
First-In-/First-Out-Speicher, dessen Daten nicht durch die Angabe einer Adresse, sondern durch die Reihenfolge der Lesevorgänge spezifiziert werden; jeder Lesebefehl holt das Datum aus dem Speicher, das sich am längsten darin befindet

Hexzahl
Die hexadezimalen Zahlen basieren auf den Potenzen der Zahl 16; die verschiedenen Stellenwerte werden durch die Ziffern 0 bis 9 und die Buchstaben A bis F dargestellt

Hohe Auflösung
Die Anzahl der Bildpunkte einer Flächeneinheit am Bildschirm entscheidet maßgeblich über die Schärfe und Deutlichkeit der Grafiken; um so mehr Punkte darstellbar sind, desto besser das Bild

LIFO-Speicher
Last-In-/First-Out-Speicher; die zuletzt gespeicherten Daten werden zuerst ausgegeben

UHF-Modulator
Eine große Anzahl von Heimcomputern kann an gewöhnliche Fernsehgeräte angeschlossen werden; dies ist jedoch nur durch sogenannte UHF-Modulatoren möglich; diese bereiten die elektronischen Signale auf, so daß sie in das TV-Gerät über die Antennenbuchse eingegeben werden können

Vorzeichensymbol
(sign) Symbol, das zur Trennung und Unterscheidung von negativen und positiven Werten dient

Zugriffsart
Der Zugriff auf gespeicherte Daten ist jeweils abhängig von dem verwendeten Speichermedium (Cassettenrecorder oder Diskettenstation); RAM-Halbleiterspeicher erlauben einen direkten Zugriff, während Magnetbandspeicher nur einen sequentiellen Zugriff erlauben

Bildnachweise:
253: Ian McKinnel
254: MTV
255: Gary Marsh, BBC Stills
259: Chris Stevens
260: Ian McKinnel/Pilot Software
261: BL Systems, Roy Ingram
263: Tony Lodge
266: Ian McKinnel
267: Trevor Hill
274: Sirius
275, 276, 279: Liz Dixon
277: Alan Coode
280: Mark Watkinson

Fachwörter auf einen Blick

Array
Bereich von Datenfeldern, die gleiche Eigenschaften haben

Bitfehler
Übertragungsfehler, die dadurch entstehen, daß ein oder mehrere Bits von 0 auf 1 oder von 1 auf 0 „umkippen"

Exklusiv-Oder-Gatter
Diese Gatter werden eingesetzt, um Prüfbits zu erzeugen; typisch ist: bei Eingabe von 0 und 0 ist die Ausgabe 0, bei Eingabe von 0 und 1 ergibt sich 1 und bei Eingabe von 1 und 1 ist die Ausgabe wiederum 0

Nodes
Der Arbeitsspeicher ist bei LOGO in Bereiche zu je 5 Bytes unterteilt; diese Bereiche heißen auch Nodes

Optokoppler
(Optoisolator) Baustein, der Daten auf Lichtstrahlen moduliert; durch den Einsatz eines Optokopplers kann erreicht werden, daß ein Datenverarbeitungssystem optisch zwar mit der Datenquelle gekoppelt, elektronisch jedoch von der Datenquelle isoliert ist

Operations-Code
Ein Teil der Anweisung einer Maschinensprache oder Assembler, in welchem die auszuführende Operation gekennzeichnet wird

PEARL
(Process and Experiment Automat in Realtime Language) Höhere, von BASIC abgeleitete Programmiersprache für die Automatisierung von Prozeßsteuerungen

Polarisationsfilter
Ein solcher Filter dient dazu, die regellosen, natürlichen Schwingungen des Lichts (Transversalschwingungen) zu „ordnen", so daß eine einheitliche Schwingungsrichtung (polarisiertes Licht) erzielt wird

Positionierzeit
heißt der Zeitraum, den der Lese- bzw. Schreibkopf eines Diskettenlaufwerks benötigt, um auf einer bestimmten Spur anzukommen, bevor der eigentliche Lese- bzw. Schreibvorgang beginnt

Querysprachen
ermöglichen die Kommunikation mit Datenbanken; mit ihren leicht erlernbaren Kommandos, die keine Programmierkenntnisse voraussetzen, lassen sich Daten abrufen, einfügen, ändern oder löschen

Redundanz
Bezeichnung für das Vorhandensein von weglaßbaren Elementen in einer Nachricht, die keine zusätzliche Information liefern, sondern lediglich die beabsichtigte Grundinformation stützen

Syntaxfehler
(syntax error) Fehler bei der Eingabe eines Programms, wenn die formellen Regeln nicht eingehalten wurden; beispielsweise vergessene Leerstelle oder Semikolon

Timesharing
Verschiedene Anwender arbeiten gleichzeitig über ein Netzwerk am selben Zentralcomputer; dabei wird abwechselnd allen Teilnehmern Verarbeitungszeit zur Verfügung gestellt; aufgrund der hohen Arbeitsgeschwindigkeit des Rechners entsteht für den Benutzer der Eindruck, daß der Computer ohne Pause nur für ihn allein tätig ist

Turtle
(englisch für Schildkröte), auch Igel genannt; kleine Bildschirmfigur bei LOGO-Grafiken; sie bewegt sich über den Screen und zeichnet dabei ihren Weg

Virtueller Speicher
nur fiktiv vorhanden; durch automatischen Austausch von Daten zwischen Arbeitsspeicher und externem Massenspeicher beim Programmablauf befinden sich im Arbeitsspeicher immer nur die Daten, die gerade verarbeitet werden; so erscheint die Speicherkapazität größer zu sein, als sie tatsächlich ist

Warteschlange
Daten, die in einer geordneten Reihenfolge auf ihre Verarbeitung in der Zentraleinheit warten

Zeichenfolge
(string) Kette von codierten alphanumerischen Zeichen

Zweidimensionales Datenfeld
Die Daten werden nach zwei unterschiedlichen Kriterien geordnet und zueinander in Beziehung gesetzt; das Feld besteht aus mindestens einer Spalte und einer Zeile, die sich kreuzen; das Feld dehnt sich in zwei Richtungen („Dimensionen") aus

Bildnachweise
282, 284, 293, 300, 305: Ian McKinnell
282: Bob Freeman
285: Tony Lodge
286: Zeff/Whelan
287, 305: Chris Stevens
295: Kai Choi
295, 303: Liz Dixon
302, 306: Kevin Jones
304: Image Maker
Fachwörter: David Higham

Fachwörter auf einen Blick

Antwortzeit
Zeitspanne zwischen dem Senden einer Anfrage bei einer Datenstation und dem Empfang einer Antwort bei derselben Station

BAM
Block Avalability Map oder Belegungstabelle; Liste der freien Sektoren auf einer Diskette

Emulationsprogramm
Programme, die auf einem bestimmten Rechner lauffähig sind, werden mit Hilfe eines Emulationsprogramms einem anderen Computer angepaßt, so daß sie auch dort arbeiten können

Hilfskanal
Bestimmte Modems für serielle Datenübertragung sehen einen Hilfskanal in der Gegenrichtung zum Datenkanal vor; er hat im allgemeinen eine wesentlich geringere Übertragungsgeschwindigkeit als der Datenkanal

Identifikation
Ein Zeichen oder eine Zeichenfolge, mittels der eine Datenendeinrichtung ihre eigene Identität kenntlich macht, auch Password genannt

Mainframe
Englische Bezeichnung für Großrechner

Mehrfachbetätigung
(rollover) Gleichzeitiges Drücken von zwei oder mehr Tasten

Multiplexer
Gerät, das mehrere Nachrichtenkanäle auf einen oder wenige Kanäle höherer Geschwindigkeit zusammenfaßt, wobei jeder eingehenden Leitung ein entsprechender Anteil der abgehenden Leitung ständig reserviert ist

Notation
Schreib- und Darstellungsweise von Zahlen in der Computertechnik; die am meisten verwendeten Notationen sind binäre, dezimale und hexadezimale; gemeint ist die Schreib- bzw. Darstellungsweise im jeweiligen Zahlensystem

OCR-Leser
Ist die Abkürzung für Optical Character Recognition; sie werden für die maschinelle Erfassung von Standard-Schriften verwendet; OCR-Leser arbeiten zeilen- und seitenorientiert; sie finden Anwendungen bei Schecks, Kreditkarten etc.

Overlay-Technik
Gewährleistet eine ausgezeichnete Nutzung der Speicherkapazitäten durch das Aufteilen von Programmen in einzelne Segmente und den damit zusammenhängenden Überlagerungsmöglichkeiten

Password
siehe Identifikationscode

Piezoelektrisch
Auf dem von Madame Curie entdeckten Effekt beruhend: bestimmte Kristalle laden sich unter Druck elektrisch auf

PL/M
(Programming Language Microprocessor) Von der Firma Intel entwikkelte höhere Programmiersprache für Microprozessoren; PL/M ist eine Art Dialekt der Programmiersprache PL/1, die die Firma IBM aus ALGOL 60 und COBOL abgeleitet hat; PL/1 wird fast ausschließlich auf Großrechnern eingesetzt

Prompt
Auf dem Bildschirm erscheinende Information, die den Bediener zur Eingabe von Daten auffordert

Referenz-Auflistung
Liste, die von einem Übersetzerprogramm erstellt wurde und mit Befehls- und Speicherplatzangaben ausgestattet ist

RGB-Monitor
In einem RGB-Monitor werden die einzelnen Signale (rote, blaue und grüne Farben sowie Zeilen und Bildschirmsynchronisation) über eine eigene Leitung übertragen

Splitmodem
Datenübertragungseinrichtung mit Multiplexer kombiniert; erlaubt die Benutzung eines Übertragungswegs für mehrere Datenverbindungen; auch: Modem mit Aufteilung in Unterkanäle

Tetrade
Eine Gruppe von vier Bits zum Darstellen von Dezimalziffern in binärer Form

Verdichten
Bezeichnet den Vorgang, der Informationen auf kleinerem Speicherplatz darstellt; Zahlen können auf diese Weise von der ASCII-(1 Ziffer/Byte) in gepackte Darstellung (2 Ziffern/Byte) verwandelt werden

Bildnachweise
309: YRM Architects & Planners
310: Roy Ingram
311: British Airports Authority
311: The Motor Ship
312: British Telecom
313: Tony Lodge
314: Starlord
315, 317, 329, 335: Ian McKinnell
317: Rosalind Buckland
319: Gary Marsh
320, 336: Kevin Jones
321: Chris Stevens
322: Marcus Wilson-Smith
329: David Weeks
330: IBM UK Ltd.
Fachwörter: Peter Fink

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Access Time = Zugriffszeit

Darunter versteht man allgemein die Zeit für das Aufsuchen einer bestimmten Information innerhalb einer großen Datensammlung. Meist ist speziell der Zugriff auf einen einzelnen Datenblock in einer Datei gemeint, vor allem bei den großen Datenbanksystemen.

Die Zugriffszeit ist etwas anderes als die „Datenübertragungsrate" – das ist die Geschwindigkeit, mit der nach der Lokalisierung die Information zwischen Speicher und Rechner übertragen wird. Beim Sinclair Microdrive beispielsweise beträgt die Zugriffszeit circa 3,5 Sek.: Sie kann auf Null reduziert werden, wenn die gesuchte Information bei Abruf gerade am Lesekopf anliegt, und maximal muß man sieben Sekunden warten, bis das Endlosband einmal ganz umgelaufen ist. Verglichen mit der üblichen Disketten-Zugriffszeit (unter 1/2 Sek.) ist das langsam; dagegen ist die Datenübertragungsrate des Microdrive mit 16 KByte pro Sekunde nicht kleiner als bei anderen Floppylaufwerken.

Accumulator = Akkumulator

Jeder Microprozessor enthält eine Reihe von „Registern" – das sind Ein-Byte-Speicher, über die alle arithmetischen und logischen Operationen abgewickelt werden. Das wichtigste und meistbenutzte Register ist der Akkumulator (kurz Akku), der direkt am Rechenwerk hängt. Die Hauptaufgabe des Akku ist die Bereithaltung von Werten und die Aufnahme von Ergebnissen. Man kann beispielsweise irgendein Byte zum Akku-Inhalt addieren oder davon abziehen. Der BASIC-Anwender hat keinen direkten Zugang zum Akku; dagegen bezieht sich bei Programmierungen im Maschinencode die Mehrzahl der Befehle unmittelbar darauf.

Acoustic Coupler = Akustikkoppler

Das öffentliche Fernsprechnetz ist so ausgelegt, daß eine Informationsübertragung nur bei Sinusfrequenzen von etwa 300–3400 Hertz möglich ist, weil das für eine verständliche Sprachwiedergabe ausreicht. Diese „Bandbreite" bestimmt auch den für digitalen Datenverkehr nutzbaren Frequenzbereich. Man benötigt daher eine Einrichtung zur Umsetzung der binären Information in geeignete Frequenzen (diese Umsetzung nennt man „Modulation"). Man kann beispielsweise der dualen Null einen Ton bestimmter Frequenz zuordnen und die duale Eins durch einen anderen Ton darstellen.

Ein Gerät zur Umwandlung der Digitalsignale in Tonfrequenzen und umgekehrt heißt „Modem" (MOdulator-DEModulator). Das Modem wird am besten direkt an die Telefonlei-

tung angeschlossen; das ist aber nur bei stationärem Betrieb möglich. Für den mobilen Einsatz benötigt man einen Akustikkoppler: Ein Modem mit zwei Gumminäpfen, in die Ohr- und Sprechmuschel des Telefonhörers eingesteckt werden. Nähme man während der Übertragung den Hörer ab, dann könnte man den Informationsfluß als Tonfolge hören.

Acronym = Akronym

BASIC ist ein Akronym, genau wie PET oder FIFO, EPROM, RAM und EMUF. Ein solches „Initialwort" wird aus den Anfangsbuchstaben der Wörter einer Bezeichnung gebildet (z. B. EMUF = Einplatinen-Mikrocomputer, Universell Festprogrammierbar). Akronyme sind in der Computerbranche sehr verbreitet, u. a. als Herstellerbezeichnungen, wobei oft der Verdacht naheliegt, daß erst das Akronym da war und nachträglich der Text dazu erfunden wurde – wie käme man sonst wohl auf Konstruktionen wie „Beginners All-purpose Symbolic Instruction Code".

ADA = ADA

Ende der siebziger Jahre wurde von der Firma Honeywell-Bull eine neue Programmiersprache konzipiert – hauptsächlich für das Pentagon, wo sie die dortige Sprachvielfalt weitestgehend maschinenunabhängig ablösen sollte. Die Sprache ist sehr stark strukturiert. Die Namensgebung erfolgte zu Ehren der Gräfin Ada Lovelace, einer engen Mitarbeiterin des englischen Rechner-Konstrukteurs Charles Babbage.

Bildnachweise

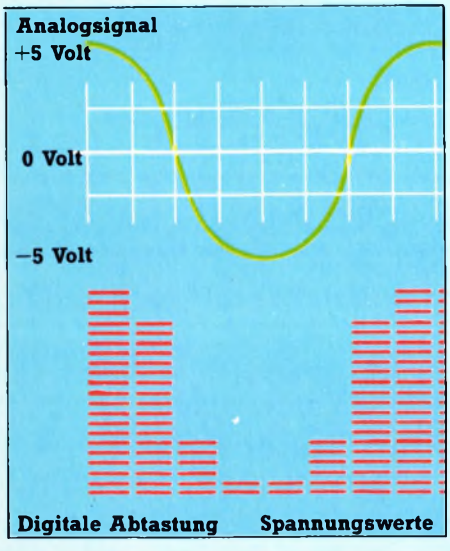
337: Columbia Warner
338: B. F. I. Stills
339: BBC Stills
339: United Artists
340, 341, 364: Ian McKinnell
348: Siemens-Museum, München
348: The Imperial War Museum
349: Chris Stevens
353: Brian Morris
354: Marcus Wilson-Smith
355: John Drysdale/Colorific
357, 358, 359: Liz Dixon
360: Roy Ingram
361: Martin Burke
362: Kevin Jones

Fachwörter von A bis Z

A/D-Convertor =
A/D-Wandler
Ein Analog/Digital-Wandler oder ADC (für A/D-Convertor) ist eine Schaltung, die analoge elektrische Signale in Digitalwerte umsetzt. Umgekehrt arbeitet ein Digital/Analog-Wandler (DAC), wozu natürlich eine ganz andere Schaltung nötig ist.

Um zu verstehen, wofür man solche Wandler braucht, müssen Sie die Begriffe „Analog" und „Digital" sorgfältig gegeneinander abgrenzen. Ein digitales Signal hat zwei Eigenschaften: Erstens ist es „diskret", d. h., es kann nur die Werte einer vorgegebenen Stufenskala annehmen – in Ihrem Computer z. B. meist nur die Werte 0 Volt und 5 Volt, entsprechend der logischen Null bzw. der logischen Eins. Zweitens ist ein Digitalsignal fast immer verschlüsselt: Es wird durch eine Folge von diskreten Werten gebildet. Acht Bits stellen im Binärsystem bekanntlich die Zahlen 0 bis 255 dar.

Ein Analogsignal ist dagegen keine diskrete, sondern eine „kontinuierliche" Variable: Sie kann innerhalb vorgegebener Grenzen unendlich viele beliebige Zwischenwerte annehmen. Außerdem ist ein solches Signal immer einer Meßgröße „analog" (daher der Name); ein Thermoelement gibt z. B. eine Spannung ab, die der gemessenen Temperatur proportional ist, und ein Mikrofon liefert eine Wechselspannung.



Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Adder =
Addierer
Ein Addierer ist eine Logikschaltung aus einfachen Verknüpfungselementen (UND-, ODER- und NICHT-Gattern), mit denen die Summe aus zwei Dualzahlen gebildet wird.

Die einfachste Version des Addierers ist der „Halbaddierer": Er hat zwei Ein-Bit-Eingänge (A und B) und zwei Ein-Bit-Ausgänge, nämlich Summe und Übertrag (carry). Ein Übertrag tritt zum Beispiel auf, wenn zugleich A und B den Binärwert 1 darstellen. In diesem Fall trägt die Summe den Wert 0; der Übertrag lautet 1.

Ein „Volladdierer" verfügt über einen zusätzlichen Eingang. Daher kann man aus Volladdierern eine Kette bilden, indem man den Übertrag eines jeden mit dem dritten Eingang (Carry-Eingang) seines linken Nachbarn verbindet. Mit acht Volladdierern lassen sich dann z. B. zwei Acht-Bit-Binärzahlen summieren (wobei das Ergebnis infolge des letzten Übertrags neunstellig werden kann).

Address =
Adresse
Speicheradressen brauchen Sie kaum, wenn Sie nur in BASIC programmieren, für den Maschinencode jedoch ist die Kenntnis der exakten Adresse Voraussetzung. Die Rechner-Zentraleinheit (CPU) kann mit einer endlichen Anzahl von Speicherplätzen (je ein Byte) kommunizieren, genannt „Adreßbereich" (beim 8 Bit-Rechner reicht er von 0 bis 65535).

Jedes Byte hat eine eigene „Rufnummer" oder Adresse, die von der CPU angewählt wird, wenn der Speicherinhalt gelesen oder überschrieben werden soll. Wird etwa das Byte Nr. 47339 benötigt, gibt die CPU diese Adresse in Binärform auf 16 parallele Leitungen, den „Adreß-Bus". Das gewünschte Byte wird dann auf den „Daten-Bus" (8 weitere Parallel-Leitungen) geschaltet, von dem die CPU die Information in ihre Register übernimmt.

ADSR = ADSR
Schon bei den ersten Heimcomputern gab es Möglichkeiten zur Tonerzeugung, bei etwas besseren Geräten auch eine softwaremäßige Steuerung der Lautstärke. Mit ADSR-Hüllkurvengeneratoren, die in anspruchsvollen Heimcomputern zu finden sind, kann der Programmierer neben Frequenz und Lautstärke auch den Klangcharakter der erzeugten Töne beeinflussen.

Vollständig heißt es „Attack-Decay-Sustain-Release-Hüllkurvensteuerung", und gemeint ist damit, daß das Klangvolumen eines Tones während seiner Erzeugung verändert werden kann. Die Struktur der Hüllkurve wird mit vier Werten definiert: Einschwingphase (Attack = Anschlag), Abklingen (= Decay), Haltephase (= Sustain; konstante Lautstärke) und Ausklingen (Release = Loslassen).

Bildnachweise
365: Ian McKinnell
366: Ian McKinnell, IO Research
367: Siggraph, Applicon, Intergraph
368, 369: Chris Stevens
370, 371: Ian McKinnell
372: Liz Dixon
378, 379: David Weeks
381: Ian McKinnell
382, 383: Kevin Jones, Ministry of Transport
384: Sperry LTD.
388: Liz Dixon
389: Kevin Jones
390: Ian Dobbie
391: Ian McKinnell, Ian Dobbie
392: Steve Cross

Fachwörter von A bis Z

ALGOL
Der Heimcomputer-Benutzer hat heute eine verhältnismäßig große Auswahl an Programmiersprachen zur Verfügung, während man früher, als es praktisch nur Mainframes (Großrechner) gab, auf FORTRAN, ALGOL und COBOL angewiesen war. FORTRAN war eigentlich für Ingenieure gedacht, COBOL für Kaufleute; ALGOL (ALGOrithmic Language) dagegen war die Sprache der Naturwissenschaftler, Mathematiker und Informatiker. Daher ist ALGOL an Hochschulen auch heute noch eine der gängigsten Sprachen. Im Microcomputer-Bereich hat ALGOL jedoch kaum Verbreitung gefunden und ist nur für die wenigsten preisgünstigen Rechner verfügbar.

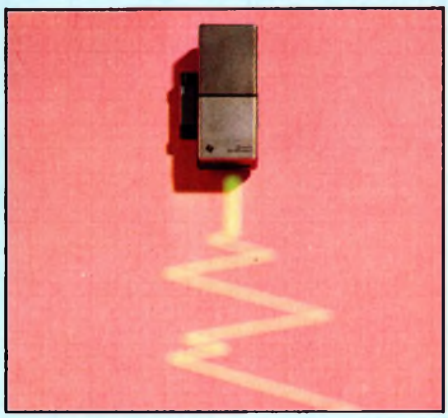
Die Stärke dieser höheren Programmiersprache liegt in der Unterprogramm-Technik: ALGOL ist ähnlich hochstrukturiert wie PASCAL und bietet dem Anwender ebenfalls eine große Anzahl fest definierter mathematisch-wissenschaftlicher Funktionen.

Algorithm = Algorithmus
Ein Algorithmus ist ganz allgemein ein mathematisches Verfahren zur Lösung eines bestimmten Problems. Algorithmen werden vorwiegend beim Programmieren eingesetzt. Manchmal besteht die ganze „Kunst" nur aus der Umsetzung eines bekannten Algorithmus (etwa für das Wurzelziehen) in die jeweilige Programmiersprache. Häufig aber liegt das Hauptproblem darin, erst einmal einen Algorithmus zur Lösung eines bestimmten mathematischen Problems zu entwickeln. Nach welchem Algorithmus finden Sie beispielsweise den Ausweg aus einem Labyrinth, oder wie muß der Algorithmus für die realistische Simulation eines Landeanflugs aussehen?

Für viele Aufgabenstellungen kann man leicht einen beliebigen Lösungsweg finden. Entscheidend ist aber, den besten ausfindig zu machen. Der eine Algorithmus führt mit einem Minimum an Speicherbedarf zum Ergebnis, ein anderer ist dafür wesentlich schneller.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Allophones = Allophone
Allophone sind bei der Sprachsynthese besonders wichtig. Die kleinsten Elemente der Sprache, die nicht weiter zerlegbar sind, heißen Phoneme; sie ermöglichen die Identifikation von Wörtern. „Reh" und „Zeh" unterscheiden sich beispielsweise durch die Phoneme „r" und



„z". Ein Phonem kann als Sprechlaut in mehreren Varianten auftreten, die man als „Allophone" bezeichnet (vgl. das „e" in „her" und „Herr"). Eine Sprache, die aus Allophonen erzeugt wird, klingt etwas künstlich, weil die Übergänge fehlen, ist aber leicht programmierbar. Zur Vereinfachung kann man bei einigen Sprachsynthese-Geräten Worte und Sätze in der üblichen Schreibweise über die Tastatur eingeben, das dann in die entsprechenden Allophone umgesetzt wird. Probleme tauchen dann auf, wenn die richtige Aussprache erst aus dem Zusammenhang hervorgeht – Beispiel: „Statt noch zu rasten, rasten sie weiter." Die Sprachsynthese anhand von Phonemen und Allophonen heißt „Synthese nach Regeln".

Das zweite Verfahren ist die „Synthese durch Analyse". Dabei spricht man zunächst die Worte ins Mikrofon, wobei der Rechner die Laute digitalisiert, die Frequenzen der Töne analysiert und in komprimierter Form abspeichert. Die Worte können dann durch umgekehrtes Vorgehen rekonstruiert werden. Dieses Verfahren ist so genau, daß man den Sprecher wiedererkennt. Hier wird das verfügbare Vokabular allerdings durch die Speicherkapazität begrenzt.

Alphanumeric = Alphanumerisch
„Alpha" steht für die Buchstaben des Alphabets, und „numerisch" für die Ziffern von 0 bis 9; alphanumerisch bedeutet also „in Buchstaben und Zahlen ausgedruckt". Gewöhnlich sind auch die Satzzeichen und einige Tastatur-Sonderzeichen mit eingeschlossen. Nicht inbegriffen sind Grafik-Symbole und verschiedene nicht ausdruckbare Zeichen wie „Wagenrücklauf", „Klingel" und „Bildschirmlöschung". Wenn das Wort „alphanumerisch" in der Funktionsbeschreibung eines Software-Pakets auftaucht, kann das vorteilhaft, aber auch stark einschränkend sein. Ein Lagerverwaltungsprogramm, das alphanumerische Zeichen (und nicht nur Zahlen) für jeden Artikel zuläßt, ist durchaus hilfreich, während Sie bei einem Drukker mit alphanumerischem Zeichensatz einen Programmausdruck erhalten, auf dem alle Grafik-Symbole fehlen.

Bildnachweise
393: Rex Features Ltd.
394: David W. Hamilton
396: University of Manchester
397, 398, 399, 400: Chris Stevens
407: Philips, Steve Cross
411: Tony Lodge
412, 417: Kevin Jones
413, U3: Ian McKinnell
415: Bob Venables
418: Steve Cross
419: Liz Dixon

Fachwörter von A bis Z

ALU = ALU
Die **A**rithmetisch **L**ogische Einheit (**U**nit) ist das Rechenwerk sämtlicher Computer. Die ALU ist Bestandteil der Rechner-Zentraleinheit (**C**entral **P**rocessing **U**nit = CPU), das sind bei Heimcomputern meist die Bausteine 6502 (Rockwell), Z80 (Zilog) oder 68000 (Motorola).

Wie der Name sagt, führt die ALU arithmetische und logische Operationen aus, z. B. Addition und Subtraktion sowie die Bedingungsabfragen mit UND, ODER, NICHT, aber jeweils nur mit Einzel-Bytes. Während die übrige CPU eigentlich nur damit zu tun hat, Daten zur rechten Zeit an den richtigen Ort zu transportieren, werden sie in der ALU tatsächlich mathematisch-logisch verändert.

AND = UND
Das AND ist eine der grundlegenden logischen Funktionen, die Ihnen sicher schon vertraut sind. AND verknüpft jeweils zwei logische Variablen. Das Ergebnis wird als „wahr" oder „falsch" ausgegeben (im Dualsystem durch 1 bzw. 0 darstellbar). Nur wenn beide Variablen zugleich „wahr" sind, ist auch das Ergebnis der Operation wahr.

Ein AND-Gatter ist eine Schaltung, mit der die AND-Funktion elektronisch realisiert wird. Ihr Heimcomputer enthält davon zahllose, und zwar integriert in größere Bausteine, beispielsweise die CPU. Aber auch beim Programmieren in der Maschinensprache gibt es meist einen AND-Befehl, der den Akkumulator-Inhalt mit einem beliebigen andern Speicherbyte verknüpft. Im Ergebnisbyte erhält eine Binärstelle genau dann den Wert 1, wenn die Operanden in der gleichen Stelle beide den Wert 1 aufweisen.

In den meisten BASIC-Versionen ist das AND ähnlich verwendbar; durch die Anweisung

```
LET C= A AND B
```

wird das Ergebnis der Variablen von A und B der Größe C zugewiesen. Ist etwa A= 13 (1101) und B= 11 (1011), dann erhält C den Wert 9 (1001). Solche Operationen sind oft sehr nützlich, z. B. bewirkt die Anweisung

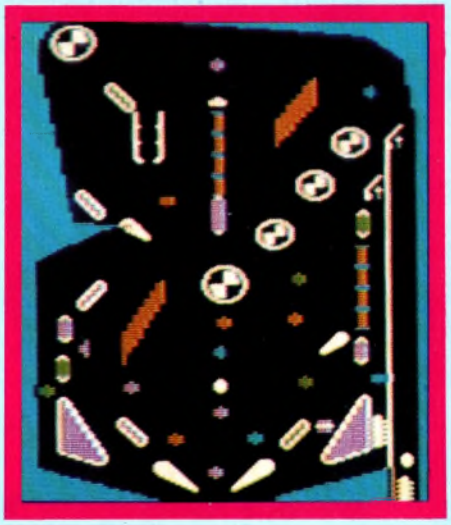
```
IF (ZEIGER AND 1) = 1 THEN
GOTO 1000
```

dann einen Sprung auf Zeile 1000, wenn die Variable ZEIGER eine ungerade Binärzahl enthält.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

APL = APL
Im Computer-Jargon gibt es eine Menge eigenwilliger Akronyme – dieses ist kaum zu übertreffen: APL steht für **A P**rogramming **L**anguage = Eine Programmiersprache! Im Widerspruch zu der schlichten Bezeichnung ist APL eine hochentwickelte Programmiersprache, die es ermöglicht, in einer Zeile APL soviel unterzubringen wie in Dutzenden von BASIC-Zeilen. Trotzdem hat APL eigentlich nur bei Mathematikern und in Hochschulkreisen Anhänger gefunden und ist für Kleinrechner zumeist nicht verfügbar. Ein anderer Grund für die geringe Verbreitung liegt darin, daß bei dieser Sprache einige Sonderzeichen verwendet werden, die es bei den wenigsten Rechnern gibt.

Geeignet für APL ist beispielsweise der Super-PET (eine Variante des Commodore 8000), der einen anderen Prozessor, ein größeres RAM und einen Zeichengenerator besitzt.



Applications generator = Anwendungsgenerator
Ein Entwicklungssystem, das Sie nur mit den Leistungsvorgaben eines Benutzers füttern müßten, damit es Ihnen selbsttätig das gewünschte Programm erstellt, ist heute schon, zumindest teilweise, realisierbar. Was zur Zeit jedoch als „Programm-Generator" verkauft wird, genügt nur bescheidenen Ansprüchen, und die erzeugten Programme benötigen wesentlich mehr Rechnerzeit und Speicherplätze als vergleichbare „handgeschriebene".

Ein „Anwendungsgenerator" hat da bessere Chancen. Die damit erzeugten Programme sind nicht selbständig, sondern laufen nur in Verbindung mit dem Entwicklungssystem. Dafür gibt es maßgeschneiderte Programme, die aus effizienten, ursprünglich handgeschriebenen Modulen aufgebaut sind.

Das Verfahren beruht darauf, daß innerhalb eines bestimmten Anwendungsbereichs (seien es nun Abenteuerspiele oder Buchhaltungsprogramme) allen Programmen gewisse Grundfunktionen gemeinsam sind, die im Entwicklungssystem „vorgefertigt" bereitstehen.

Bildnachweise
421: Ian Dobbie
422: Anni Axworthy
423, 430, 431, 440, 441, 443, U3: Ian McKinnell
432, 442, 445, 446, 447: Kevin Jones
439: Liz Dixon

Fachwörter von A bis Z

Architecture = Architektur
In der Umgangssprache beinhaltet der Begriff „Architektur" nicht nur die äußere Gestaltung eines Bauwerks, sondern auch seine funktionale Ordnung. Bei Rechnern wird dieser Ausdruck ausschließlich für die interne Struktur gebraucht. Unter der „Architektur" eines Microprozessor-Bausteins versteht man meist den logischen Aufbau der internen Register und ihre Arbeitsweise. Es besteht jedoch auch die Möglichkeit, daß die spezielle Halbleitertechnologie des Bausteins gemeint ist. Wird der Begriff dagegen für den ganzen Rechner angewandt, bezieht er sich auf die Art und Anordnung der Systembausteine und die Aufteilung der verfügbaren Speicherkapazität. Bei Rechnern mit der gleichen Architektur ist eine weitgehende Kompatibilität gegeben.

Array = Datenfeld
In einem „Array" werden Daten, die zueinander in Beziehung stehen, in systematischer Anordnung, etwa in Form einer Tabelle, mit Zeilen und Spalten abgelegt. Handelt es sich beispielsweise um eine zweidimensionale Tabelle, brauchen Sie zur eindeutigen Spezifizierung der Daten zwei Kennzahlen, nämlich den Zeilen- und den Spaltenindex. Ihr Rechner akzeptiert auch ein eindimensionales Datenfeld – das entspricht dann einer (einspaltigen) Liste. In BASIC werden zum Beispiel Dimension und Umfang eines Feldes durch eine DIMension-Anweisung spezifiziert.

Nichtmathematikern fällt das Verständnis für Felder mit mehr als drei Dimensionen schwer. Für den Rechner ist es aber nur eine Frage des Betriebssystems und des Speichers, ob er nun zwei oder 13 Dimensionen (d. h. 13 Indizes) verarbeiten kann.

Das Wort „Array" wird manchmal auch bei der Hardware benutzt, und zwar meist in der Verbindung „Arrayprozessor". Dieser enthält eine Serie konventioneller Rechenbausteine, so daß Rechenoperationen mit großen Datenmengen stark beschleunigt werden können.

Artificial Intelligence = Künstliche Intelligenz
Viele Menschen sehen in der Vorstellung von einer „intelligenten Maschine" einen inneren Widerspruch, weil sie einem Apparat, der nur eingegebene Daten nach einem von Menschen entwickelten Schema (Programm) verarbeitet, keine Kreativität zutrauen.

Es gibt aber seit einiger Zeit Programme mit einer gewissen Lernfähigkeit – z. B. Schachprogramme, die nach zahllosen Spielen soviel „Erfahrung" gewinnen, daß sie sogar den Programmierer mattsetzen. Andere „intelligente" Programme führen selbständige Korrekturen aus, so daß das Listing nach einigen Durchläufen nicht wiederzuerkennen ist. Ob es sich nun tatsächlich um Intelligenz handelt, sei dahingestellt; immerhin werden an Hochschulen Forschungen unter dem Titel „Künstliche Intelligenz" durchgeführt.

Der englische Mathematiker Alan Turing hat in den fünfziger Jahren folgenden „Intelligenztest" für Computer vorgeschlagen: Sie setzen eine Person vor zwei Fernschreiber, von denen einer mit einem (versteckten) menschlichen Partner, der andere mit dem Test-Gerät in Verbindung steht. Wenn der Prüfer an seinen Fernschreibern durch keine noch so raffinierte Frage herausbekäme, wann der menschliche Partner und wann die Maschine antwortet, dann wäre dem Rechner Intelligenz nicht abzusprechen.

Für die meisten Forschungsergebnisse auf dem Gebiet der KI gibt es nur begrenzte Anwendungen. Dazu gehören Sprach- und Bildmuster-Erkennung und sogenannte Expertensysteme, bei denen versucht wird, Spezialkenntnisse zu simulieren.

Bei der KI gibt es zwei gegenläufige Vorgehensweisen: Beim „top-down"-Verfahren (von oben nach unten stufenweise Verfeinerung; „Baumstruktur") versucht man, Denkprozesse wie den Umgang mit Sprache auf normalen Rechnern nachzuahmen. Die „bottom-up"-Anhänger dagegen beginnen mit den Details. Sie versuchen, ein elektronisches Äquivalent zu den Gehirn-Nervenzellen (Neuronen) zu entwickeln. Im Verbund großer Netzwerke zeigen solche Elemente sogar schon Ansätze, aus Umgebungserfahrungen zu lernen.

Alan Turing (1912–1954)

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Bildnachweise
449: Paul Chave
450: Tony Sleep
451: Steve Cross
452: Royal Aeronautical Society
453, 469, 470, 471: Ian McKinnell
454, 455: Chris Stevens
457, 462, 472, 473, 476: Kevin Jones
463: David Weeks

Fachwörter von A bis Z

ASCII = ASCII
Der „American Standard Code for Information Interchange" ist von den meisten Hard- und Softwareanbietern für die Binär-Darstellung alphanumerischer Zeichen übernommen worden. Es gibt zwar keinen besonderen Grund, den Buchstaben A ausgerechnet als „0100 0001" (8-Bit-ASCII-Norm) zu codieren, aber die Vereinbarung einer Norm bietet den Vorteil, daß Programme auf unterschiedlichen Maschinen verwendbar werden. So wird eine problemlose Datenübertragung von einem zum anderen Rechner möglich. Kommerzielle Software-Pakete für Kleinrechner bieten oft die Möglichkeit, ASCII-Dateien zu erzeugen, deren Sinn am besten ein Beispiel erläutert: Bei der Textverarbeitung enthalten die abgespeicherten Schriftsatz-Dateien meist eine Anzahl spezieller Steuerzeichen, etwa um die Zentrierung der Überschrift auf einer Seite zu kennzeichnen. Macht der Rechner daraus eine ASCII-Datei, so setzt er überall statt der Steuerzeichen ASCII-Symbole ein, hier also entsprechend viele Leerzeichen. Die ASCII-Datei kann daher auch von einem zweiten Textverarbeitungssystem mit einem anderen Steuerzeichen-Satz gelesen werden.

Assembly Language = Assemblersprache
Der Assemblercode ist im wesentlichen nur eine lesbarere Schreibweise der Maschinensprache, mit „symbolischen" (alphanumerischen) anstelle hexadezimaler Adressen und mnemotechnischem Befehlscode statt eines Zahlenschlüssels. Im Gegensatz zu anderen höheren Sprachen werden die Daten bei der Assembler-Übersetzung Byte für Byte in den Maschinencode übertragen. Daher sind Assemblerprogramme ebenso effizient wie Maschinenprogramme. Das vom Benutzer geschriebene Quellprogramm, auch Source Code genannt, wird durch einen Übersetzer (der selbst „Assembler" heißt) im Rechner in ein Objektprogramm im Maschinencode umgewandelt.

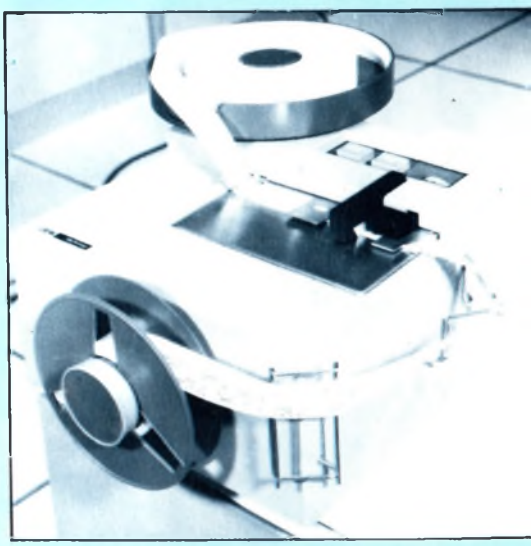
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Asynchronous = Asynchron
Daten können synchron oder asynchron übertragen werden. Meist werden diese Begriffe verwendet, wenn es um die serielle Datenfernübertragung geht, zum Beispiel bei Telefonleitungen. Die asynchrone Übertragung ist einfacher durchführbar und über die gängigen Schnittstellen, wie RS 232, zu bewerkstelligen. Die synchrone Übertragung erlaubt jedoch höhere Datenraten und wird daher bei allen großen Rechnersystemen eingesetzt.

Beim Asynchron-Verfahren beginnt die Übertragung, wann immer das Sendegerät Daten ausgeben will. Jedem Byte wird ein zusätzliches „Start-Bit" vorangestellt, mit dem das empfangende Gerät darauf hingewiesen wird, daß gleich acht Informationsbits eintreffen, und zwar mit der Baudrate, auf die Sender und Empfänger eingestellt sind. Mit dem „Stop-Bit" wird die Übertragung des Bytes abgeschlossen. In dieser Abfolge wird ein Byte nach dem anderen übertragen.

Beim Synchronverfahren haben Sender und Empfänger präzise synchronisierte Taktgeber, und die Übertragung erfolgt in einem festgelegten Rhythmus. Es ist etwa so, als ob Sie sagen: „Ich warte zu jeder vollen Stunde fünf Minuten lang am Telefon auf deinen Anruf." – wobei der Rechner allerdings nur Mikrosekunden warten würde. Das Synchronverfahren benötigt keine Start- und Stop-Bits und arbeitet entsprechend schneller.

Attribute = Attribut
Die ersten Computer waren nur in der Lage, normale Zeichen darzustellen und allenfalls zwischen Klein- und Großbuchstaben zu unterscheiden. Heute können Sie schon bei einfachen Heimcomputern für jeden Buchstaben einzelne Farben festlegen, dazu noch die des Hintergrundes, außerdem Fett- oder Negativschrift wählen. Diese Zusatzangaben heißen „Attribute". Meist ist für jede Zeichenposition des Schirms ein Byte im RAM vorgesehen. Das Setzen der Attribute erfolgt entweder durch BASIC-Befehle wie BRIGHT, FLASH, INK, PAPER usw. oder durch unmittelbare Bitmanipulation im Maschinencode über AND/OR-Verknüpfungen.



Das System der Verschlüsselung läßt sich anhand eines Lochstreifengerätes veranschaulichen: Jedes ASCII-Zeichen wird durch eine Reihe von maximal sieben oder acht Löchern dargestellt; zwischen der dritten und der vierten Lochposition ist noch ein kleines Führungsloch für den Streifentransport.

Bildnachweis
477: The Kobal Collection
480: Ian McKinnell, Soft
481: Kevin Jones, Your Spectrum
482, 483: Kevin Jones
484: Liz Dixon, Ian McKinnell
485, 500, 501: Ian McKinnell
490, 491, 495: Tony Lodge
499: Alan Adler
504: Adrian Morgan
U3: IBM

Fachwörter von A bis Z

Background = Hintergrund
Ein professionelles Textverarbeitungssystem belegt – selbst bei kontinuierlichem Schreiben – nur etwa zehn Prozent des Leistungsvermögens Ihres Computers. Während Sie kaum mehr als zehn Anschläge pro Sekunde schaffen, könnte der Rechner das Zehnfache bewältigen (Tastaturabfrage, Zeicheninterpretation, RAM-Speicherung und Bildschirmwiedergabe). Deshalb hängt der Rechner rund 90 Prozent der Zeit nur in einer Warteschleife, bis Sie die nächste Taste drücken.

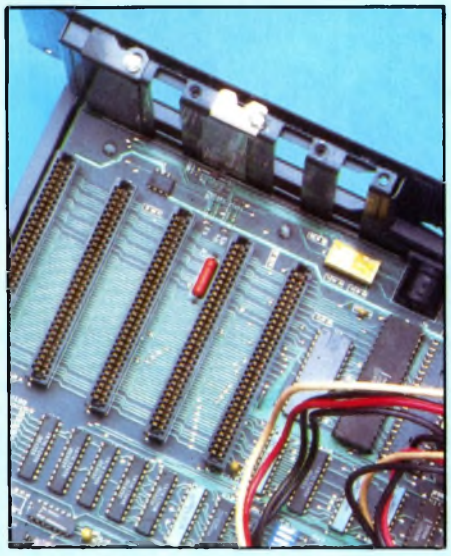
Die Auslastung des Computers wäre erheblich besser, wenn die Zentraleinheit in den Wartepausen andere Aufgaben erledigen könnte. – Darauf basiert die „Hintergrund"-Programmierung: Der Rechner arbeitet im „Vordergrund" (foreground) im Dialog mit dem Benutzer, während im Hintergrund ein anderes Programm abläuft.

Dies bedeutet aber auch, daß nur bestimmte Aufgaben für den Hintergrund geeignet sind. Beispielsweise erfordern Sortierprogramme viel Rechenzeit, obwohl die ständige Mitwirkung des Benutzers nicht notwendig ist. So auch beim Druckvorgang: Im Hintergrund wird ein Dokument ausgedruckt, während Sie schon am nächsten Text arbeiten.

Zeitintensive Aufgaben wie das Sortieren laufen im Hintergrund nicht ganz so schnell ab wie bei völlig freiem Rechner, denn die meisten Vorder/Hintergrund-Systeme arbeiten nach dem Zeitanteilverfahren (Timesharing): Der Prozessor verwendet beispielsweise 1/100 Sekunde auf den Vordergrund, dann 1/100 Sekunde auf den Hintergrund usw. Bei Großrechnern werden auf diese Weise im „Multiprogramming" Hunderte von Benutzern zugleich bedient.

Nur bei wenigen Kleinrechnern ist das Standard-Betriebssystem für diese Arbeitsweise eingerichtet. Meist müssen Sie dazu spezielle Software (unter Namen wie „Multitasking" – oder „Concurrent"-System) kaufen, beispielsweise die Concurrent-Version des bekannten CP/M-Systems.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



Backplane = Rückwandplatine
Bei den meisten Heimcomputern sind heute alle Bausteine in einem geschlossenen Gehäuse untergebracht. Die ersten Kleinrechner sahen aber ganz anders aus. – Der Rechner hatte hinten eine frei zugängliche „Rückwandplatine" mit einer Anzahl Steckerleisten für den Anschluß von Leiterplatten und Peripheriegeräten.

Der Vorteil dieses Konzepts bestand nicht nur in der leichten Erweiterbarkeit (beispielsweise durch Speicher- oder Schnittstellenkarten), sondern auch in der Möglichkeit, den Computer mehr oder weniger selbst ausbauen zu können.

Einiges von diesem System ist bei modernen Ausbildungs- und Bürorechnern übernommen worden. Beim „Tycom Microframe" ist man noch einen Schritt weitergegangen und überläßt dem Benutzer sogar die Wahl des Prozessors, ebenso die des RAM und der Schnittstellen.

Alle Anschlußstecker für die Rückwandplatine müssen die gleiche Stiftbelegung für die Bus-Kopplung aufweisen. Bei den ersten Maschinen war die „S-100"-Norm (mit 100poligem Stecker) sehr verbreitet.

Backup = (Sicherungs-) Duplikat
Die Bedeutung der Datensicherung bei Rechnern kann man nicht genug betonen. Unter einem „Backup" versteht man eine Reservekopie von einem Datenträger, die für den Fall der Beschädigung des Originals angefertigt wird. Solche Kopien sind bei Magnetträgern (Disketten und Cassetten) unerläßlich, weil trotz der verbesserten Zuverlässigkeit damit immer Pannen passieren können. Die Diskette kann verlorengehen, versehentlich mit Kaffee begossen oder auf einen starken Magnet gelegt werden, oder schlicht „dropouts" (Zeichenausfälle) bekommen, wenn sich mikroskopisch kleine Teilchen von der Magnetschicht lösen.

Bekanntlich können winzige Fehler oder Störungen eine ganze Datei wertlos machen. Wie oft Sie bei der Programmentwicklung regelmäßig Duplikate anlegen (wofür im Betriebssystem eine spezielle Routine vorgesehen ist), sollte davon abhängen, wie tragisch für Sie ein Verlust der betreffenden Datei wäre. Die meisten lernen das spätestens durch schmerzliche Erfahrungen, wenn der Rechner eine in vielen Stunden erstellte Datei plötzlich nicht wieder verarbeiten „will".

Bildnachweise
505, 517, 529: Steve Cross
506: Kevin Jones
507, 508, 509, 518, 519, 527, U3: Chris Stevens
514, 530: Liz Heaney
515: Liz Dixon
516: Tony Sleep, Microscope
522: Rolf Seiffe
524: Steve Malone
526: July Goldhill
528: Ian McKinnell

Fachwörter von A bis Z

Bandwidth = Bandbreite
Der Begriff „Bandbreite" wird meist in nachrichtentechnischem Zusammenhang gebraucht. Die Bandbreite eines Übertragungskanals (z. B. verdrillte Leitung, Telefonkabel, Laserstrahl oder Funkverbindung) ist der Frequenzbereich, der bei der Übermittlung ohne nennenswerte Signalverzerrung nutzbar ist. Das Telefon arbeitet beispielsweise mit dem Tonfrequenzbereich von 300-3400 Hz. Die Bandbreite beträgt 3,1 kHz. Das genügt für Sprachübertragung, aber nicht für HIFI-Musik.

Die Bandbreite der Verbindung bestimmt die maximale Datenrate beim Verkehr zwischen den einzelnen Geräten. Ein Fernsehkanal belegt einen Frequenzbereich, der für 3000 parallele Ferngespräche ausreichte. Während ein preisgünstiges Telefonmodem eine Datenrate von nur etwa 1200 Baud bewältigt, bietet der Computer die Möglichkeit, bei Verwendung des Kabelfernsehnetzes in Sekundenbruchteilen ein längeres Programm an einen anderen Benutzer übertragen.

Bank Switching = Bankauswahlverfahren
Jeder Microprozessor hat einen vorgegebenen Adreßbereich, der die Anzahl der identifizierbaren Speicherplätze bestimmt. 8-Bit-Zentraleinheiten wie die verbreiteten Bausteine Z80 oder 6502 können gewöhnlich einen Adreßbereich von 64 KByte (Speicherplätze 0 bis 65535) ansprechen. Die neueren 16-Bit-Rechner (auch der Sinclair QL) sind in der Lage, sehr viel mehr Speicherraum zu adressieren, zum Teil mehrere Megabyte.

Eine Speichererweiterung über 64 KByte hinaus ermöglicht bei einem 8-Bit-Prozessor nur das sogenannte „Bank Switching", eine gängige Programmiertechnik bei Rechnern wie dem Commodore 64 oder dem Atari. Obwohl dem Rechner eine festgelegte Speicherkapazität zur Verfügung steht, läßt sich mit Hilfe des Betriebssystems, durch ein internes Umschalten (Switching), der Speicherplatz doppelt nutzen. Ähnlich

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

funktioniert auch das „paging", wobei der Rechner zum Beispiel bei der Bildwiedergabe veranlaßt wird, je nach Wunsch etwa den halben oder den doppelten Bereich des Bildschirm-RAMs für die Darstellung zu nutzen. Das Bank Switching können Sie sich so vorstellen: Die „memory map" des Computers ist eine senkrechte Liste. Zeilenweise werden nun RAM-„Streifen" hin- und hergeschoben, bis sich die benötigten zusätzlichen Speicherbereiche in der „memory map" befinden.

Bar Codes = Strichcodes
Strichcodes kennt wohl jeder – und sei es nur von Verpackungsetiketten.

Strichcodes werden nicht nur auf Preisschildern verwendet. Bei dieser Orgel von Casio sind die Noten auf dem Papier als Strichcodes verschlüsselt und können mit einem Lichtstift in den Speicher eingelesen werden.

Diese Form der Verschlüsselung von Digitalwerten ist mit Hilfe eines optischen Lesestiftes elektronisch lesbar. – Er wird einfach darüber hinweg geführt.

Strichcodes werden aber nicht nur bei Preis- und Produkt-Etiketten verwendet. Sie sind z. B. auch zur Noteneingabe bei Synthesizern (wie bei den Casio-Orgeln) zu gebrauchen, und Hewlett-Packard verwendete Strichcodes erstmals zum Programmeinlesen vom Papier in den Rechner.

Schwierigkeiten bereiten bei der Abtastung der Strichcodes die verschiedenen Geschwindigkeiten, in denen der Lesestift über die Codes geführt wird.

Base = Basis
Beim vertrauten Dezimalsystem wird meist vergessen, daß es ein „Stellenwertsystem" darstellt. Dessen „Basis" 10 ist jedoch für die Rechnertechnik weniger geeignet als die Basiszahlen 2 (Dual- oder Binärsystem) oder 16 (Hexadezimal-, kurz: Hex-System).

Die Basis eines Zahlen-Systems kann beliebig groß sein; bei der Zeitangabe in Minuten und Sekunden z. B. wird die Basis 60 benutzt (Sexagesimal-System). Wird die Basis 10 überschritten, müssen zur Ergänzung der Ziffern 0–9 neue Symbole gefunden werden – im Hex-System werden die „Ziffern" 10–15 durch die Buchstaben A–F dargestellt. Während sich im normalen Gebrauch das Dezimalsystem durchsetzen konnte, arbeiten Rechner im Dualsystem, da elektrisch zwei stabile Zustände wesentlich besser realisierbar sind.

Bildnachweise
533: Chris Stevens
534, 538, 546, 548, 556, 557, U3: Ian McKinnell
535: Chris Stevens, Ian McKinnell
539: Marcus Willy
540, 559: Kevin Jones
541: Steve Cross
547: Michael Brownlow, Kevin Jones
549: David Weeks
551: Sir John Tenniel
553: Marcus Wilson-Smith
555: Tony Duncan Smith

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

BASIC = BASIC
Dieser Begriff ist jedem Computer-Besitzer vertraut, und die meisten wissen auch, daß dies ein Akronym von „Beginners All-purpose Symbolic Instruction Code" (Anfänger-Allzweck-Symbolsprache) darstellt. Weniger geläufig ist, daß BASIC am Dartmouth College (USA) als Sprache für den Informatikunterricht entstanden ist.

BASIC wurde von FORTRAN abgeleitet, der damals in Wissenschaft und Technik verbreitetsten Sprache. Dabei wurden umständliche Anweisungen wie zum Beispiel WRITE und FORMAT durch den einfachen PRINT-Befehl ersetzt.

Der eigentliche Fortschritt besteht aber darin, daß BASIC schon vom Konzept her eine Dialogsprache ist: Alle Eingaben erfolgen direkt durch den Benutzer am Terminal und nicht, wie früher üblich, durch Einlesen vorgestanzter Lochkarten. Programmfehler sind daher im Dialog durch Überschreiben einzelner Zeilen leicht zu korrigieren.

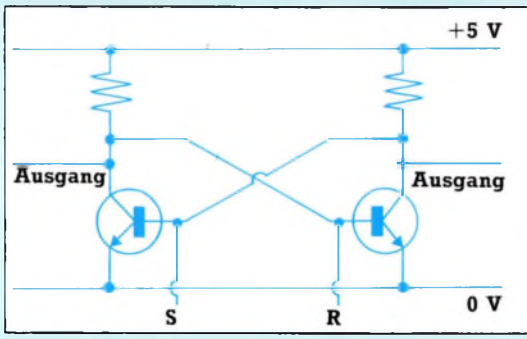
BCD = BCD
Beim BCD-Code (Binary Coded Decimal) wird eine Dezimalzahl Stelle für Stelle binär verschlüsselt, z. B. zum Abspeichern auf Diskette. Die meisten Heimcomputer arbeiten zur besseren Speichernutzung allerdings mit Fließkommadarstellung (Floating Point). Dabei wird die Dezimalzahl als Ganzes in eine Binärzahl umgewandelt und anschließend mit dem Dezimalpunkt versehen. Der Punkt wird dabei in der „Mantisse" ganz nach links geschoben und der Stellenwert in einen Exponent übernommen. Beide werden in einer festen Anzahl von Bytes abgespeichert.

Beim BCD-Code wird jede Ziffer der ursprünglichen Dezimalzahl in eine vierstellige Dualzahl (d. h. 1/2 Byte) umgesetzt, so daß insgesamt halb so viel Bytes wie dezimale Stellen erforderlich sind. Das BCD-System behandelt jede Ziffernstelle einzeln. Dagegen wird beim Fließkommaverfahren die Zahl als Ganzes verarbeitet.

Benchmark = Bewertungsstandard
Als die ersten Personal Computer wie der PET, der Apple II und der TRS-80 von Tandy auf den Markt kamen, wurde ein Bewertungssystem entwickelt, um die Verarbeitungsgeschwindigkeit und die Effizienz der BASIC-Interpreter zu vergleichen. Es bestand aus zehn einfachen Programmen, die unterschiedliche Aspekte berücksichtigten (Schleifenbearbeitung, Fließkommarechnung, Winkelfunktionen usw.). Damit erzielte Testzeiten finden Sie in technisch orientierten Computerzeitschriften, etwa in der Form „BM1–10,2 s; BM2–3,87 s" und so fort.

Es ist nicht gelungen, ein entsprechendes System für moderne Bürorechner einzuführen – hauptsächlich deshalb, weil die Effizienz solcher Anlagen entscheidend vom speziellen Zuschnitt der Programme auf den jeweiligen Rechner abhängt. Der 8-Bit-Rechner Osborne 1 z. B. gilt nicht gerade als flink, trotzdem wird er von vielen Journalisten für die Textverarbeitung bevorzugt, weil er mit dem WordStar schneller arbeitet als die meisten neuen 16-Bit-Rechner.

Bistable = Bistabile Kippstufe
Als „bistabil" bezeichnet man die sog. „Flip-Flop"-Schaltungen mit ihren zwei stabilen Zuständen. Am einfachsten ist das RS-Flip-Flop realisierbar. Der Schaltzustand des Flip-Flops wird durch die Ausgangsspannung angezeigt, die z. B. 5 V („High"-Pegel) oder 0 V („Low"-Pegel) betragen kann.



Ein RS-Flip-Flop ist eine bistabile Kippschaltung, die als „Gedächtnis" für ein einzelnes Bit dient. Mit Kombinationen von solchen Flip-Flops können Binärzahlen gespeichert werden.

Das RS-Flip-Flop eignet sich als 1-Bit-Speicher, der je nach Schaltzustand eine „1" oder eine „0" enthält. Die ersten Halbleiterspeicher bestanden aus Serien von solchen Flip-Flops, und die heute noch üblichen statischen RAMs sind dasselbe in kompakter Bauweise. Der Trend geht allerdings zu dynamischen RAMs, bei denen die Information in Form elektrischer Ladungen auf winzigen Kondensatoren gespeichert wird. Wegen des Spannungsabfalls benötigen dynamische RAMs spezielle (in die Bausteine integrierte) Auffrisch-Schaltungen. Sie sind aber schneller und brauchen weniger Strom als die statischen RAMs.

Flip-Flops sind weiterhin als logische Einzelschaltkreise zu finden. Es gibt unterschiedliche Ausführungsformen. Das oben abgebildete RS-Flip-Flop hat zwei Eingänge: Ein Impuls am S-Eingang „kippt" das Flip-Flop in den einen Zustand, während es durch einen Impuls am R-Eingang „zurückgekippt" wird.

Bildnachweise
561, 563, 564, 565, 573, 575: Ian McKinnell
566, 577: Kevin Jones
574: Chris Stevens
579: Tony Lodge
581, U3: Liz Dixon
582: Commodore, Tony Sleep
583: Apple, Osborne
584: Sinclair Research, Hewlett Packard, Judy Goldhill

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Bit = Bit
Offiziell ist dieses Wort eine Abkürzung des Begriffes Binary DigIT (Binärstelle) als Name für die kleinste Informationseinheit. Das „Bit" taucht im Computer-Jargon in vielen Zusammensetzungen auf. Hier eine kleine Auswahl:

Bit-Kopierer: Ein Hilfsprogramm, das einen Datenträger Bit für Bit auf einen anderen kopiert. Mit diesem Verfahren werden die meisten Software-Raubkopien erstellt, da so manche Schutzvorkehrung umgangen werden kann.

Bit-Fehler: Das „Umkippen" eines oder mehrerer Bits auf einen anderen logischen Status. Dabei kann es sich um nichtreproduzierbare Fehler handeln, wie beispielsweise infolge von Schaltspitzen im Stromnetz oder infolge kosmischer Strahlung während erhöhter Sonnenfleckenaktivität. Diese treten nach Neustart des Rechners nicht wieder auf. Oder es handelt sich um permanente Fehler, deren Ursache z. B. ein defekter Transistor in einem RAM-Baustein sein kann.

Bit-parallel: Ein Verfahren, bei dem alle Bits einer Informationseinheit auf entsprechend vielen parallelen Leitungen gleichzeitig übermittelt werden. Dies ist aufwendiger, aber schneller, als alle Bits nacheinander „bitseriell" auf einer einzigen Leitung zu übertragen.

Bit-Verarbeitung (auch Bit-Manipulation): Bezeichnung für die unterste Stufe der Informationsverarbeitung (nämlich bitweise).

Block = Block
Ein „Block" ist eine definierte Datenmenge, und zwar die kleinste Einheit, die bei einem Schreib- bzw. Lesevorgang gleichzeitig verarbeitet wird. Wenn bei einem Diskettenlaufwerk jeweils vier komplette Sektoren zu je 256 Bytes beschrieben werden, dann beträgt die Blocklänge 1 KByte. Jeder Block ist eine abgeschlossene Einheit und enthält oft noch Kontrollinformationen, die eine Fehlererkennung beim Einlesen ermöglichen. Daneben sind Anfang und Ende des Blocks mit besonderen Kennsymbolen ausgezeichnet.

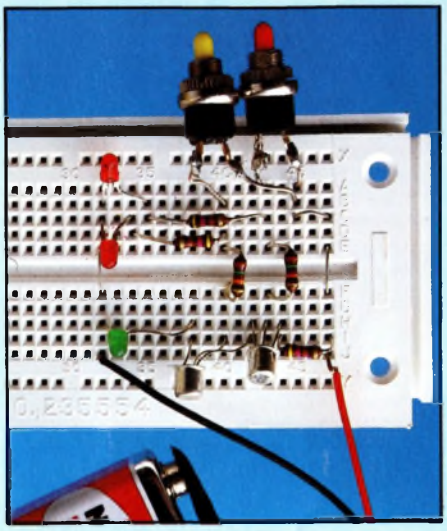
Boot = Booten
Ein Programm auf Diskette kann sich nicht selbständig in den Speicher laden. Dafür muß das Betriebssystem sorgen. Dieses befindet sich bei Heimcomputern üblicherweise im ROM und wird daher beim Einschalten des Rechners sofort aktiviert. Bei größeren Rechnern ist das Betriebssystem ein separates Programm im RAM.

Für das Einlesen ist ein im ROM abgelegtes Programm, der „bootstrap loader" oder Urlader erforderlich. Der Urlader lädt nach dem Einschalten das Betriebssystem Byte für Byte in einen bestimmten RAM-Bereich und startet es. Dieser Vorgang wird im Computer-Jargon als „Booten" bezeichnet.

Branch =
Sprung oder Verzweigung
Die Ausführung einer Sprunganweisung bewirkt, daß der Rechner das Programm nicht mit dem nächstfolgenden Befehl fortsetzt, sondern zu einem anderen Programmteil springt. Zu unterscheiden sind der „unbedingte" Sprung, der in jedem Fall erfolgt, und der „bedingte" Sprung, der an die Erfüllung bestimmter Bedingungen gebunden ist. In BASIC bewirkt der einfache GOTO-Befehl einen unbedingten Sprung auf die in der Anweisung genannte Zeilennummer. Dagegen würde die bedingte Sprunganweisung

IF SUM > 4 THEN GOTO 100

nur dann einen Sprung auf Zeile 100 auslösen, wenn SUM größer ist als 4.



Breadboard = Experimentierplatte
Als Breadboard werden Hilfsmittel bezeichnet, auf denen Elektronik-Bauteile für Versuchsaufbauten verschaltet werden können. Meist handelt es sich um gelochte Platten, die einseitig mit Lötaugen oder parallelen Kupferbahnen ausgestattet sind. IC-Sockel, Transistoren, Widerstände usw. werden mit ihren Pins durch die Karte gesteckt und verlötet. Die Leiterbahnen werden, wo nötig, mit einem Schaber aufgetrennt oder mit Draht überbrückt, um die gewünschte Schaltung zu realisieren.

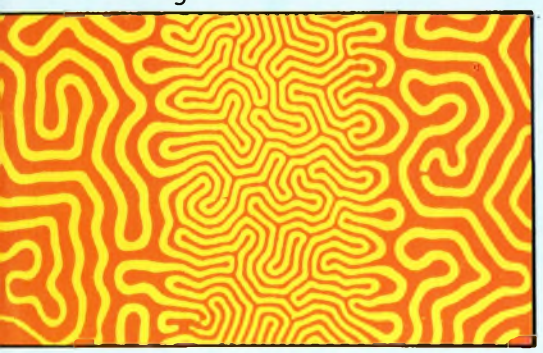
Alternativ gibt es auch lötfreie Experimentier-Steckbretter, die mehr für die Ausbildung als für den professionellen Gebrauch gedacht sind. Sie enthalten in einem zentimeterstarken Plastikgehäuse eine Vielzahl von Löchern mit intern verbundenen Federbuchsen. Die Bauelemente werden nur eingesteckt und können zur Veränderung der Schaltung leicht wieder entfernt werden.

Bildnachweise
589, 592: Steve Cross
590: Kevin Jones
593, 595: Chris Stevens
596: Sainsbury
597: Tony Duncan-Smith, Liz Dixon
604, 613: Ian McKinnell
605: Marcus Wilson-Smith
606: BBC Television
611: Liz Dixon
612: Alun Jones
614, 615: David Higham

Fachwörter von A bis Z

Break = Unterbrechung
Die Ausführung eines Programms läßt sich im allgemeinen durch Betätigung einer Sondertaste (z. B. STOP, BREAK, ESCAPE) unterbrechen. Dabei werden alle Informationen über den Stand der Programmbearbeitung in einem reservierten Speicherbereich abgelegt, zum Beispiel in welcher Zeile unterbrochen wurde oder welchen Inhalt Schleifenzähler und Unterprogrammregister hatten. Dadurch ist gewährleistet, daß das Programm ohne Datenverlust oder -veränderung wieder gestartet werden kann.

Solche Unterbrechungen sind beispielsweise bei der Fehlersuche nützlich. Sie unterbrechen an einer kritischen Stelle, schauen sich den gegenwärtigen Variablenwert an, verändern ihn bei Bedarf und starten an der gleichen Stelle wieder.



Magnetfeld-Darstellung

Bubble Memory = Blasenspeicher
In Magnetblasenspeichern wird die Information durch Magnetfelder dargestellt, also nicht durch elektrische Spannungen oder Ladungen wie beim Halbleiter-RAM. Anders als bei Magnetmedien wie Band oder Diskette gibt es beim Blasenspeicher aber keinerlei mechanische Bewegung. Informationsträger ist eine dünne Magnetschicht auf einem Granat-Kristall, in der eine Vielzahl von winzigen „Blasen" mit gegensätzlichen Magnetisierungsrichtungen (entsprechend der logischen Null bzw. Eins) gebildet wird.

Die einzelnen Blasen sind in Form von Schleifen miteinander verbunden und werden durch ein äußeres Magnetfeld in Bewegung gehalten. Diese Speicherschleifen werden über eine zentrale Zugriffsschleife sequentiell abgefragt bzw. beschrieben. Das geht zwar schneller als bei Band oder Platte, jedoch langsamer als bei direktem Speicherzugriff. Demgegenüber hat der Blasenspeicher jedoch den Vorteil, daß die Daten auch bei Stromausfall erhalten bleiben.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Bubble Sort = Bubble Sort
Der „Bubble Sort" ist ein Sortieralgorithmus, der ein Datenfeld alphabetisch oder nach der Größe ordnet. Das Verfahren ist einfach zu verstehen. Die Arbeitsweise ist jedoch vergleichsweise unwirtschaftlich. Der Name rührt daher, daß beim wiederholten Durchgehen des Feldes die Daten wie Luftblasen (bubbles) zu dem für sie bestimmten Platz im Feld aufsteigen.

Buffer = Buffer
Ein Hardware-Buffer gleicht die elektrischen Spannungsdifferenzen aus, die zwischen den verschiedenen Hardware-Komponenten des Computers entstehen. Die meisten Peripheriegeräte sind in der Lage, mehr Informationen zu verarbeiten, als der Microprozessor in einem bestimmten Zeitraum weiterleiten kann. Ein spezieller Buffer-Chip sorgt dabei für die Verstärkung des digitalen Signals.

Der Software-Buffer gleicht dagegen die Übertragungsdifferenzen zwischen dem Betriebssystem und den anderen Systemkomponenten aus. Dieser Buffer ist im allgemeinen Teil des RAM-Bereiches untergebracht. Er kann zum Beispiel von einem Peripheriegerät mittels einer entsprechend hohen Übertragungsgeschwindigkeit mit Daten gefüllt und von einem anderen, das nur eine niedrigere Rate verarbeiten kann, geleert werden. Buffer dieser Art werden zwischen Prozessor und Tastatur, Diskettenstation und Drukker eingesetzt.

Bei der Textverarbeitung können Sie die Arbeitsgeschwindigkeit mit einem großen Buffer zwischen Rechner und Drucker erheblich vergrößern. Wenn Sie den Druckbefehl geben, lädt der Rechner die fertige Seite in den Buffer. Dieser wird dann vom Drucker langsam geleert, während Sie bereits die nächste Seite eintippen, ohne warten zu müssen. Kleinere Buffereinheiten haben eine Kapazität bis zu vier KByte, was für ein Schriftstück mit 700 Wörtern ausreicht.

Bus = Bus
Ein Bus ist ein elektrischer Übertragungskanal für Informationen innerhalb eines Rechnersystems, meist in Form einer Anzahl von Leiterbahnen auf der Platine. Der Systembus ist über den Bus-Erweiterungsstecker von außen erreichbar. Intern verbinden drei wichtige Busse die CPU mit den übrigen Komponenten: Der „Adreßbus" überträgt Speicher- und Geräteadressen, der „Datenbus" die eigentlichen Informationen, und über den „Steuerbus" werden Signale zur Steuerung und Synchronisation der verschiedenen Systembausteine ausgetauscht.

Bildnachweise
617, 619, 640: Steve Cross
618, 633, 636, 637: Kevin Jones
620, 622, 623: Ian McKinnell
621: Collins/Duncan Smith
628: Acorn
629, 630, 631: Chris Stevens
638: Liz Heaney
641: Record Mirror
643: Liz Dixon
U3: New Scientist

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

CAD/CAM = CAD/CAM

Die Begriffe CAD und CAM (Computer Aided Design/Manufacturing) für rechnergestützten Entwurf bzw. rechnergestützte Fertigung charakterisieren gemeinsam ein modernes Industriekonzept. „Entwurf" heißt in diesem Zusammenhang: ingenieurmäßige und nicht künstlerische Gestaltung – obwohl Computer jetzt auch in diesem Bereich erfolgreich eingesetzt werden.

Der Rechnereinsatz in der Konstruktion bietet wesentliche Vorteile – zunächst einmal bei der bildlichen Darstellung des Entwicklungsobjekts. CAD-Systeme arbeiten meist mit einem hochauflösenden Monitor sowie einem farbfähigen Plotter. Die zugehörige Software ist praktisch das grafische Gegenstück zu einem Textverarbeitungspaket: Bildelemente können korrigiert, gedreht oder gelöscht werden, und auf der

Diskette befindet sich eine „Bibliothek" für Standardbauteile. Bei aufwendigeren Systemen ist auch die räumliche Darstellung des Entwurfs möglich, der aus verschiedenen Blickwinkeln betrachtet werden kann.

Weiterhin kann der Computer mit Hintergrund-Programmen den Entwickler von langwierigen, aber unentbehrlichen Berechnungen entlasten, wie zum Beispiel die Bestimmung der aus Festigkeitsgründen erforderlichen Materialquerschnitte oder die Erstellung von Formeln und Funktionen für die in der Zeichnung benötigten Kurven.

Schon im Vorfeld der Konstruktion kann der Rechner die Suche nach der optimalen Lösung unter gegebenen Randbedingungen (Gewicht, Größe, Stabilität und Kosten) wirksam unterstützen, indem alle Möglichkeiten durchkalkuliert werden und so das optimale Ergebnis ermittelt wird.

Von großem Vorteil ist ferner, daß mit der Vollendung des Entwurfs alle Konstruktionsdaten gespeichert vorliegen und unmittelbar an CAM-Anlagen weitergegeben werden können.

Der Begriff CAM (rechnergesteuerte Fertigung) umfaßt eigentlich den gesamten Bereich der Robotics. In der Industrie geht der Trend von starren Fertigungsstraßen weg zu flexiblen Produktionsanlagen (Flexible Manufacturing Systems = FMS), die leicht umzuprogrammieren sind und eine bessere Anpassung der Fertigung an die Marktsituation ermöglichen. Dabei werden an den Fließbändern CNC-Werkzeugmaschinen (Computer Numerically Controlled = rechnergesteuert) eingesetzt, die bei Produktionsänderungen softwaremäßig „umgerüstet" werden können.

CAI/CAL = CAI/CAL

Hier geht es um Computer und Lernprozesse – ein umstrittenes Thema. Selbst unter den Befürwortern des Rechners in Schulen gibt es noch Debatten darüber, ob der Computer besser als „Lehrhilfe" (CAI) oder als „Lernhilfe" (CAL) zu gebrauchen sei.

CAI (Computer Aided Instruction) ist die rechnergestützte „Unterweisung" im Stil des traditionellen Lehrens. Der Computer ist zugleich elektronisches Lehrbuch und rechnergesteuerte Lehrkraft. Im Rahmen eines Lehrprogramms wird der Stoff abschnittweise anhand von Grafiken, Trickfilmszenen und Sound angeboten. Das ist viel interessanter als das klassische Lehrbuch oder die Tafelmethode. Nach jedem Abschnitt versucht der Rechner (üblicherweise in Form von Multiple-Choice-Fragen), den Lernerfolg festzustellen. Das Programm ermöglicht dem Schüler, die individuelle Lerngeschwindigkeit selbst zu bestimmen.

Beim CAL (Computer Aided Learning) steht dagegen das „Lernen aus eigenem Antrieb" im Vordergrund, wovon man in der Vergangenheit aufgrund fehlender Voraussetzungen kaum gezielt Gebrauch machen konnte. Der Computer simuliert eine Umgebung, die zum „spielerischen" Lernen anregt.

Zweifellos wird CAL weit eher als CAI der kindlichen Auffassungsgabe gerecht. Das CAL-Konzept erfordert allerdings weit mehr Phantasie und auch raffiniertere Programme.

CALL = CALL

Der CALL-Befehl veranlaßt den Rechner, ein Unterprogramm aufzurufen. Wenn Sie diese Anweisung nicht in der Befehlsliste Ihres Rechners finden, gibt es dafür bestimmt etwas Gleichwertiges, beispielsweise GOSUB, PROC, SYS oder USR. Oft hat der CALL-Befehl auch die Bedeutung, daß der folgende Programmteil in der Maschinensprache geschrieben ist. Das entspricht ebenfalls dem Aufruf eines Unterprogramms (im Maschinencode).

Bildnachweise
645, 646, 647: Chris Stevens
648, 649, 652, 661, 667: Kevin Jones
650: Janos Marffy
651, 665: Ian McKinnell
660: Liz Heaney
666: Brian Morris
670: David Higham
671: Soft
672: Imagine
U3: Integraph

Fachwörter von A bis Z

Carrier Tone = Trägerfrequenz
Zum Datenaustausch über das Fernsprechnetz ist ein Akustikkoppler oder ein Modem erforderlich, weil Telefonleitungen nur einen bestimmten Frequenzbereich übertragen. Wenn Verbindung zwischen zwei Rechnern besteht, liegt ständig die „Trägerfrequenz" auf der Leitung. Diese ist als hoher Ton hörbar.

An den meisten Modems und Akustikkopplern befindet sich ein LED mit der Bezeichnung „Carrier Detect", die erlischt, wenn die Trägerfrequenz unterbrochen wird. Die Trägerfrequenz ist mitten ins Fernsprech-Übertragungsband gelegt; die Daten werden nach einem bestimmten Schema moduliert. Eine „1" könnte z. B. durch einen etwas über, eine „0" durch einen etwas unter dem Träger liegenden Ton darge-

stellt werden. Das Modulationsschema bestimmt die maximale Übertragungsrate – bei den meisten Modems nicht mehr als 1200 Baud, bei einigen Geräten aber auch bis zu 9600 Baud.

In dem Film „Wargames" dringt ein Computer-Freak in das militärische Rechnernetz der USA ein, wobei er mit einem Programm Computer ausfindig macht, die telefonisch anzuzapfen sind. Das Programm, das er entwickelt hat, wählt einfach eine Nummer nach der anderen in einem Ortsnetz. Sobald eine Trägerfrequenz übermittelt wird, speichert das Programm die Nummer auf Diskette ab. Die meisten Datenbanken sind jedoch vor unrechtmäßiger Abfrage geschützt.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Carry = Übertrag
Ohne „Übertrag" von einer Stelle zur benachbarten funktioniert das Rechnen in Stellenwertsystemen nicht – auch Ihr Computer macht beim Rechnen im Binärsystem davon Gebrauch. Eine einfache Halbaddierer-Schaltung, die zwei einstellige Binärzahlen addiert, hat z. B. außer dem Summen- noch einen Übertrags-Ausgang. Ein Volladdierer hat noch einen dritten Eingang für etwaige Überträge aus der nächstniedrigeren Ziffernstelle. Acht gekoppelte Volladdierer können zwei Acht-Bitzahlen addieren, u. U. mit neunstelligem Ergebnis. Das Rechenwerk (ALU = Arithmetic and Logical Unit) in der Zentraleinheit (CPU) Ihres Computers hat in einem solchen Fall dafür zu sorgen, daß das „Carry Bit" in einem der internen Register als Überlaufanzeige gesetzt wird.

CCD = CCD
Charge-Coupled Devices (Ladungsgekoppelte Bauelemente) ersetzen zunehmend die klassischen Bildröhren in Fernsehkameras, die Computern als optische Eingabegeräte eine Art „Gesichtssinn" verleihen können. Dies ist besonders für den Bereich der Automatisierung interessant, wo man erreichen möchte, daß Industrieroboter Bauteile identifizieren.

Verglichen mit konventionellen Fernsehkameras ist ein CCD-System kleiner, leichter und in großen Stückzahlen billiger. Als Bildempfänger dient dabei ein Raster aus z. B. 256x256 winzigen Ladungsspeichern, die auf einem Chip integriert sind. Darauf wird mit der üblichen Optik das Bild projiziert, nachdem jedes Speicherelement aufgeladen wurde (z. B. auf die „1"). Die Zellen entladen sich nun entsprechend der örtlichen Bildhelligkeit, und das Bild kann durch Ladungsübertragung ausgelesen werden – bei digitaler Auswertung wären jetzt dunkle Bereiche durch „1", helle durch „0" gekennzeichnet. Die Lage der Hell/Dunkel-Schwelle, die die Zuordnung eines Bildteils zur 1 oder 0 bestimmt, ist von der Belichtungszeit abhängig.

Cell = Tabellenfeld
Als Felder bezeichnet man bei der Tabellenkalkulation die einzelnen Sektoren im Schnittpunkt von Zeilen und Spalten. Jedes Feld ist über die Zeilen- und Spaltenkoordinaten individuell adressierbar und kann nach Wahl mit Text oder Zahlenwerten belegt werden, aber auch mit einem arithmetischen Ausdruck wie „84+82", wobei der Inhalt des Feldes durch den zweier anderer definiert wird.

Centronics = Centronics
Die amerikanische Firma Centronics hat für ihre Drucker eine Parallel-Schnittstelle eingeführt, die sich in der ganzen Branche durchgesetzt hat. Auf acht Leitungen werden die zu einem Byte gehörigen Bits gleichzeitig übertragen. Der Centronics-Standardstecker ist 36-polig, es sind aber auch kleinere Stecker verwendbar. Geräte mit Centronics-Schnittstelle sind problemlos zu koppeln. Die Schnittstelle ist jedoch nicht bidirektional, d. h. nicht für Geräte geeignet, die zugleich Sender und Empfänger sind wie z. B. Modems.

Bildnachweis
673: Ian McKinnell, Teletape Video Ltd.
674, 680, 685, 689, 693, 696, 697: Ian McKinnell
675, 686, 687: Steve Cross
678, 679: Ian McKinnell, CL Projects
681, 682, 690: Kevin Jones
684: Dave Cooper-Smith
691: Ian McKinnell, Kevin Jones
695: Liz Heaney
698, 699: Ian McKinnell, Commotion Ltd.

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Chain = Kette
Eine Kette ist eine bestimmte Ordnungsstruktur für Datensätze. Bei der Verkettung enthält jeder Datensatz einen Zeiger mit der Adresse des nächsten Satzes. Wie bei jeder anderen Kette hat aber auch hier der Bruch eines einzigen Gliedes (etwa ein Speicherfehler) entscheidende Folgen: den Verlust der gesamten Datei, weshalb meist auch noch alternative Zugriffsmöglichkeiten vorgesehen werden.

Die meisten Rechner können auch vollständige Programme verketten: Der Rechner lädt ein erstes Programm, läßt es ablaufen, lädt automatisch das nächste usw. Dies wirkt nach außen wie der Ablauf eines einzigen langen Programms. Bei Computerspielen wird beispielsweise oft zuerst nur die Spielanweisung geladen, die anschließend mit dem Hauptprogramm überschrieben werden kann. Auf diese Weise steht für das Hauptprogramm der ganze Speicher zur Verfügung.

Der Begriff „Daisy-Chain" kennzeichnet dagegen eine Hardware-Konfiguration, und zwar meist die Art des Anschlusses der Peripherie an den Computer. Eine Möglichkeit besteht beispielsweise in der Verwendung einer speziellen Rechner-Schnittstelle für jedes Gerät, eine andere in der Verbindung aller Komponenten durch ein gemeinsames Bus-System. Bei der Daisy-Chain wird an die Peripheriebuchse des Rechners ein Gerät angeschlossen, das seinerseits wieder über ein entsprechendes Interface verfügt, so daß dort ein zweites Gerät eingesteckt werden kann usw.

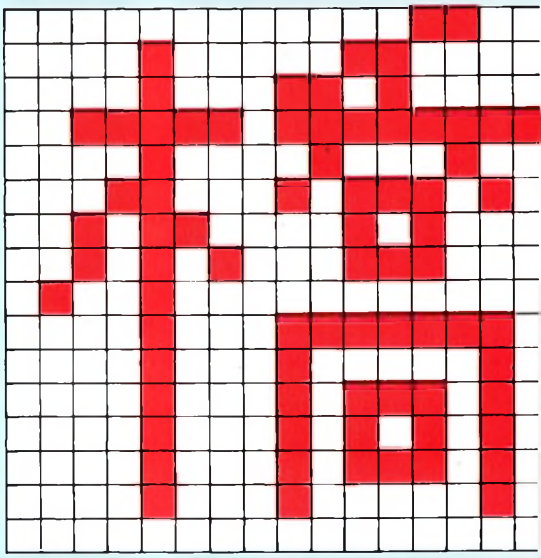
Channel = Kanal
Ein Kanal ist ein Transportmedium für Daten. Bei Computern ist damit nicht die Hardware (etwa eine Schnittstelle oder Leitung) gemeint, sondern das zugehörige Programm für die Datenflußsteuerung. Bei den meisten Betriebssystemen ist eine Ausgabe über den Bildschirm, den Drucker oder auf Diskette erst möglich, wenn der richtige Kanal geöffnet ist. Anschließend werden die Daten an den zugeordneten Kanal gesendet, der durch einen Namen oder eine Nummer gekennzeichnet ist, und das Betriebssystem leitet sie zu dem entsprechenden Gerät.

Die softwaremäßige Organisation von Kanälen hat auch den Vorteil, daß der Benutzer die Geräte nur indirekt spezifizieren muß. Wenn in den Ausgabeanweisungen eines Programms etwa über die Kanalnummer 5 der Drucker angesprochen wird und die Ausgabe aber auf den Plotter umgeleitet werden soll, brauchen Sie beim Programmstart nur anzugeben, daß über Kanal 5 jetzt der Plotter bedient wird, und müssen nicht die entsprechenden PRINT- und WRITE-Anweisungen einzeln ändern.

Character Generator = Zeichengenerator
Der Zeichengenerator ist für die Erzeugung der einzelnen Symbole zuständig. Üblicherweise ist im ROM für jedes Zeichen das zugehörige Punktmuster in Form einer Folge von Einsen und Nullen abgelegt.

Bei den meisten Heimcomputern läßt sich die Speicherung der Zeichenmuster aber auch in einen bestimmten RAM-Bereich übertragen. Dabei können Sie die Gestalt jedes alphanumerischen Zeichens oder Grafiksymbols beliebig verändern und sich so einen maßgeschneiderten Zeichensatz schaffen.

Check Digit/Check Bit = Prüfziffer/Prüfbit
Zur Fehlererkennung bei der Übertragung und Speicherung von Informationen dienen zusätzliche Prüfziffern oder Prüfbits. Solche Prüfzeichen werden vor allem bei der Datenfernübertragung oder beim Speichern auf Magnetträgern verwendet, weil dabei sehr leicht Fehler auftreten können.



Ein Zeichengenerator erzeugt Textzeichen und andere Symbole in Form eines Punktrasters. Bei einigen Rechnern können Sie das Punktmuster selbst definieren, wie am Beispiel des japanischen Schriftzeichens für „Brücke" gezeigt.

Prüfziffern sind Zusatzinformationen, die aus den Daten nach einer bestimmten mathematischen Vorgabe erzeugt und mit übertragen werden. Beim Datenempfang (oder Lesen des Speichermediums) wird nach der gleichen Regel verfahren. – Entspricht das Ergebnis nicht dem ursprünglichen Prüfzeichen, hat sich ein Fehler eingeschlichen.

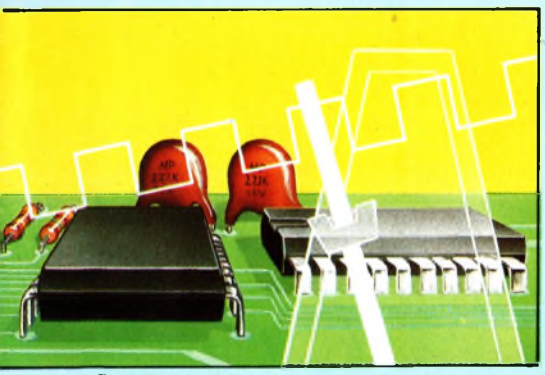
Die Erzeugung des Prüfzeichens kann mehr oder weniger aufwendig gestaltet werden. Sie kann beispielsweise so erfolgen, daß man alle Bytes in einem Datenblock addiert und aus dem Divisionsrest beim Teilen durch 256 ein Prüfbit bildet (Prüfsummen-Verfahren).

Bildnachweis
701: Steve Cross
702, 703, 707, 722: Kevin Jones
705: The Science Museum
711, 719, 720: Ian McKinnell
713, 714, 715, 724, 725: Chris Stevens
716: Bob Hall
717: Roy Ingrams
721: Marcus Wilson-Smith
U3: Liz Dixon

Fachwörter von A bis Z

Clock = Taktgeber
Der Taktgeber im Computer hat die undankbare Aufgabe, den Rechner langsamer zu machen. Die Befehlsausführung erfolgt in der CPU in elementaren Einzelschritten durch Transistor-Schaltungen. Dabei erfordert jede Operation eine gewisse Schaltzeit, um die unterschiedlichen Transistorzustände umzusetzen.

Die Verarbeitungszeiten variieren bei den verschiedenen Gattern, und mit wachsender Anzahl der beteiligten Logikstufen kann der Datenfluß außer Takt geraten. Wenn bei einem Prozessor, der acht Bit parallel verarbeitet, das letzte Bit an einem Addierer-Eingang erst nach der Summation der übrigen erscheint, ist das Ergebnis natürlich wertlos.



So wie ein Metronom dem Musiker gibt die Clock der Zentraleinheit des Rechners den Takt an.

Die ganze Rechnerlogik einschließlich der Speichersteuerung muß daher durch einen zentralen Taktgeber synchronisiert werden. Erst der Taktimpuls veranlaßt die Informationsübernahme durch die nächste Stufe. Die Taktintervalle müssen länger sein als die Ansprechzeit des langsamsten Schaltkreises.

Als Taktgeber dient ein Quarzkristall, der mit einer bestimmten Frequenz von einigen Megahertz „schwingt". Microprozessoren arbeiten vielfach mit Taktfrequenzen von ein, zwei, vier oder sogar acht Megahertz.

Die Taktfrequenz wird von Herstellern häufig als Maß für die Rechengeschwindigkeit hingestellt, was aber allenfalls für den Vergleich von Rechnern mit identischem Prozessor sinnvoll ist. Ein Rechner mit 6502-Prozessor und 2-MHz-Takt ist nicht unmittelbar einem Rechner mit Z80-Prozessor bei 2 MHz gegenüberzustellen. Abgesehen davon spielen noch andere Einflüsse eine Rolle, so daß die Taktfrequenz höchstens einen vagen Anhaltspunkt für die Rechengeschwindigkeit gibt.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

CMOS = CMOS
Bei der Halbleiter-Herstellung werden in das Halbleitermaterial gezielt Fremdatome eingebaut (Dotierung), die je nach Art eine „n-Leitung" durch negative Elektronen oder eine „p-Leitung" durch positive Ladungsträger („Löcher" in der Elektronenbesetzung) bewirken. Die Anordnung von p- und n-leitenden Zonen im einem Halbleiterkristall bestimmt die Funktion des Bauteils: Eine Diode beispielsweise besteht nur aus einer p- und einer angrenzenden n-Schicht, während ein „bipolarer" Transistor drei Zonen mit der Schichtfolge p-n-p oder auch n-p-n enthält.

Bipolar-Transistoren benötigen einen erheblichen Steuerstrom, weil ihre Steuerelektrode (die „Basis") niederohmig ist. Beim Feld-Effekt-Transistor (Fet) kann man dagegen die Steuerelektrode (aus Metall) durch eine Oxidschicht vom Halbleiter (Semiconductor) isolieren. Ein solcher „MOS-Fet" ist fast leistungslos steuerbar. Je nach Dotierung des Halbleitermaterials spricht man dabei von einer PMOS- oder NMOS-Technologie.

Die gleichzeitige Verwendung von PMOS- und NMOS-Fets wird als CMOS (complementary MOS) bezeichnet, weil dabei jedes Gatter komplementär aus spiegelbildlichen PMOS- und NMOS-Hälften aufgebaut ist. Da immer eine davon gesperrt ist, ergibt sich ein extrem niedriger Energieverbrauch – ideal für batterieversorgte Geräte wie beispielsweise Taschenrechner. Zum Beispiel ist der CMOS-bestückte Tandy Modell 100 netzunabhängig, und der Speicherinhalt bleibt sogar wochenlang erhalten.

Coaxial Cable = Koaxialkabel
Mit schlichten Leitungsdrähten oder Flachbandkabeln lassen sich zwar niederfrequente Signale übertragen, aber für höhere Frequenzen sind sie nicht geeignet. Störsignale aus der Umgebung, Impulsverzerrungen und Dämpfung machen das Signal mit der Leitungslänge zunehmend unbrauchbar. Kabel mit einem koaxial in einem geerdeten Abschirmschlauch aus Metallgeflecht angeordneten isolierten Innenleiter sind dagegen hochfrequenzgeeignet und beispielsweise für die Verbindung von Rechner und Bildschirm unerläßlich. Koaxialkabel werden auch beim Aufbau von Netzwerken zur Verbindung der einzelnen Stationen untereinander verwendet.

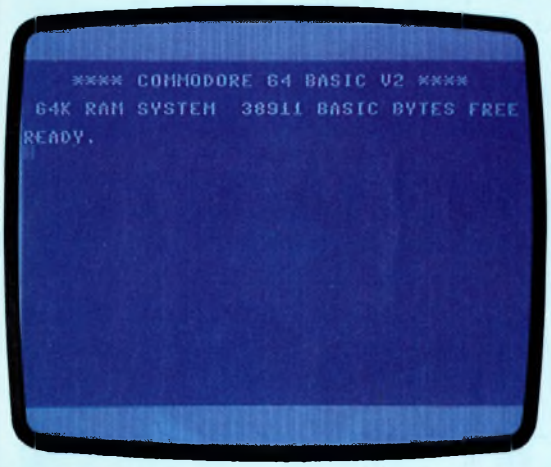
COBOL = COBOL
COBOL (COmmon Business Orientated Language=Universalsprache für kaufmännische Anwendungen) war die erste Programmiersprache, die nicht für mathematisch-naturwissenschaftliche Zwecke entwickelt wurde. Sie hat vor allem im kommerziellen Bereich bei Großrechenanlagen und Rechnern der mittleren Datentechnik Verbreitung gefunden. COBOL ist aber auch für Rechner mit CP/M-Betriebssystem verfügbar.

Bildnachweis
729: Ian McKinnell, Syco Systems
730: Mary Evans Picture Library, Bernhard Jennings
736, 744, 754: Ian McKinnell
737, 742, 743: Chris Stevens
739: Liz Dixon
747: Steve Cross
748: William Stuart Systems Ltd.
749: Newcastle Polytechnic
755: Atari

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Cold Start = Kaltstart
Ob Ihr Rechner nun eine Sekunde oder tagelang abgeschaltet war – beim Einschalten gibt es jedesmal einen „Kaltstart". Dabei dauert es Sekundenbruchteile, bis alle Schaltkreise einen stabilen Betriebszustand erreicht haben. Über ein simples Zeitglied (Kondensator mit Ladewiderstand) wird etwa eine Zehntelsekunde nach dem Einschalten ein Impuls auf den Reset-Eingang des Prozessors gegeben, der sich in der Zwischenzeit stabilisiert hat.



So meldet der C 64, daß er betriebsbereit ist und genügend Bytes frei hat.

Der Reset-Impuls sorgt dafür, daß der Prozessor die „Startschwierigkeiten" überwindet und auf eine bestimmte ROM-Adresse springt, mit der das „Kaltstartprogramm" beginnt. Dieses durchläuft zunächst einen RAM-Funktionstest, fragt den verfügbaren Speicherraum ab und startet dann das Betriebssystem. Einen „Warmstart" veranlassen Sie dagegen durch Drücken der Reset-Taste. Dabei werden die Prozessor-Register zurückgesetzt, in manchen Fällen werden auch der Speicherinhalt gelöscht und das Betriebssystem aufgerufen.

Command Language = Kommandosprache
Sie kennen den Unterschied zwischen menü- und befehlsgesteuerten Programmen: Beim Menü wählen Sie durch einen Tastendruck aus einem Angebot von Varianten, während Sie bei der Befehlssteuerung ein Kommandowort eintippen müssen. Die Menütechnik zeichnet sich durch Anwenderfreundlichkeit aus. Bei Verfügbarkeit einer „Kommandosprache" läßt sich allerdings mit der Befehlssteuerung wesentlich mehr anfangen.

Eine Kommandosprache ermöglicht dem Benutzer die Kombination mehrerer Einzelaktionen in einem Befehlswort. Bei einer Datenbank brauchen Sie beispielsweise häufig Befehlsfolgen wie: GET = Hole den nächsten Datensatz, EXTRACT = entnimm das Feld TOTAL, MULTIPL(Y)iziere es mit dem DISCOUNT = Rabatt-Satz, UPDATE = aktualisiere den Datensatz mit dem neuen Wert und STORE = speichere ihn wieder in der Datei. Bei einer Kommandosprache reicht für diese Vorgänge ein Einzelbefehl, etwa MODIFY.

Hochentwickelte Datenbanksysteme wie dBaseII von Ashton-Tate gehen noch einen Schritt weiter: Dort können Sie ganze Programme mit Verzweigungen und Unterprogrammen in der Kommandosprache schreiben.

Bildnachweise

757, 758, 759: Andy Leslie
757, 779: Mike Brownlow
758, 763, 767, 778, 783: Ian McKinnell
759, 773, 780: Kevin Jones
760, 761, 769, 770, 771: Chris Stevens
763, 765, 786, 775, 776: Liz Dixon
774: Chris Stevens, Texas Instruments
779: Mike Brownlow

Comparator = Komparator
Ein Komparator ist ein Schaltkreis, der zwei Analogspannungen miteinander vergleicht und ein digitales Ausgangssignal abgibt.

Ein Komparator wird verwendet, um einen Analog/Digital-Wandler (AD) mit Hilfe eines Digital/Analog-Wandlers (DA) aufzubauen. Ein DA ist nämlich viel leichter realisierbar als ein AD. Der Ausgang des DA kommt an den einen Komparator-Eingang, die Meßspannung an den anderen. Über eine Steuerlogik mit Speicherregister wird an den DA eine binäre Zahlenfolge gelegt, die eine schrittweise Zunahme seiner Ausgangsspannung bewirkt. Erreicht sie die Höhe der Meßspannung, so „kippt" der Komparator „um" und hält mit seinem Ausgangssignal die Steuerlogik an – im Speicherregister steht dann der digitalisierte Meßwert zur Verfügung.

Compiler = Compiler
Ein Compiler ist ein Übersetzungsprogramm, das in einer höheren Programmiersprache geschriebene Programme oder „Quellprogramme" in den Maschinencode (in „Objektprogramme") umsetzt.

In einer Compilersprache entwikkelte Programme sind wesentlich schneller als die im üblichen BASIC geschriebenen. Die meisten BASIC-Versionen arbeiten nämlich nur als „Interpreter", das heißt, sie übersetzen die Anweisungen während des Programmlaufs nacheinander in Maschinenbefehle. Zum Beispiel wird eine Schleife bei jedem Durchlauf neu vom BASIC in den Maschinencode übersetzt.

Ein Compiler entschlüsselt dagegen die Anweisungen nur einmal vor der ersten Programmbenutzung und speichert sie als Maschinenprogramm ab. Während des Programmlaufs fällt dann keine Übersetzungszeit mehr an. Für Heimcomputer werden verhältnismäßig selten Compiler angeboten, weil sie sehr viel Speicherraum beanspruchen und eine dialogorientierte Programmentwicklung dabei nur unter Vorschaltung eines Interpreters möglich ist.

Fachwörter von A bis Z

Complement = Komplement
Das Komplement einer einstelligen Zahl ist die Ziffer, die sie zur Basis des Zahlensystems ergänzt. Zum Beispiel ist im Dezimalsystem (Basis 10) die 7 das Komplement zur 3. Mit Hilfe des Komplements läßt sich die Subtraktion auf eine Addition zurückführen. Die Aufgabe (8 − 3 = ?) läßt sich auch so formulieren:
(8 + (10 − 3)) − 10 = ?
Das ist einfacher als es aussieht:
10 − 3 = 7 ist das Komplement von 3, und statt 10 von der Summe 8 + 7 = 15 zu subtrahieren, brauchen Sie nur den bei der Addition entstehenden Zehner-Übertrag zu unterschlagen.

Im Binärsystem ist zur Bildung des erforderlichen Komplements lediglich eine Vertauschung von Nullen und Einsen vorzunehmen (beispielsweise entsteht aus 1011 das „Einerkomplement" 0100) und dann eine Eins zu addieren (damit wird aus 0100 folglich das „Zweierkomplement" 0101).

Concatenate = Verketten
Der Begriff „Verkettung" wird bei der Programmierung meist im Zusammenhang mit alphanumerischen Strings oder mit Diskettendateien verwendet. Die Strings A$ und B$ können zum Beispiel durch die Anweisung C$ = A$ + B$ zu einem neuen String C$ verkettet werden. Wenn zwei oder mehr Dateien miteinander verbunden werden, entsteht daraus unter einem neuen Namen eine große Gesamtdatei.

Concurrency = Mehrprogrammbetrieb
Ein „Concurrent"-Betriebssystem ermöglicht die gleichzeitige Abwicklung mehrerer Programme mit nur einem Rechner. Die Bearbeitung erfolgt dabei nicht simultan, sondern „verzahnt" – der Prozessor schaltet ständig zwischen den verschiedenen Aufgaben hin und her. Dieses „Multiprogramming"-Verfahren wird bei Großrechnern seit Jahren angewendet, während es bei kleineren Computern, aufgrund der niedrigen Verarbeitungsgeschwindigkeit, noch nicht weiter verbreitet ist. Unter dem

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Das Betriebssystem „Concurrent CP/M" ist für viele Bürorechner verfügbar, so auch für den IBM-PC. Ein 16-Bit-Prozessor, ein großer Arbeitsspeicher und schnelle Diskettenlaufwerke sind Voraussetzung für die parallele Abwicklung von mehreren Programmen.

Betriebssystem „Concurrent CP/M" von Digital Research laufen bis zu vier CP/M-86-Programme parallel.

Bei nur einer Tastatur, einem Bildschirm und einem Operateur müssen drei der Programme selbständig ablaufen. Über das Tastenfeld ist allerdings jedes der Programme zu starten, abzubrechen oder auf dem Bildschirm darzustellen. Beim Concurrent CP/M können mit Hilfe des „Piping"-Verfahrens auch die Ergebnisse des einen Programms automatisch als Eingabewerte dem nächsten zugeleitet werden. So kann ein Datenbankprogramm bestimmte Informationen heraussuchen und an ein Textverarbeitungssystem weitergeben, von dem ein drittes Programm den Bericht zum Ausdrucken im Hintergrund übernimmt.

Constant = Konstante
Unter dem Namen einer Variablen werden im Rechner Speicherplätze verwaltet, deren Inhalt sich während des Programmlaufs ändern kann, während eine numerische oder eine String-Konstante unveränderlich ist. In der Anweisung:
A = B* 6.5731
sind A und B Variablen, 6.5731 ist eine Konstante.

Konstanten weisen gegenüber Variablen einige Nachteile auf. Erstens kosten Konstanten unnötige Rechenzeit, weil fast alle Rechner mit Gleitkommaarithmetik arbeiten und nicht im BCD-System. In einer Programmschleife wird daher in der obigen Anweisung die Konstante bei jedem Durchlauf vom BASIC-Interpreter erneut in Gleitkommadarstellung umgewandelt, während Variable bereits richtig gespeichert vorliegen.

Außerdem belegt die mehrfache Verwendung einer Konstanten in einem Programm unnötig viel Speicherplatz. Vorteilhafter ist die Verwendung von Variablen, denen Sie zu Beginn des Programms den Wert der Konstanten zuweisen. Wenn Sie alle Konstanten-Zuweisungen im Programmkopf als Block zusammenfassen, sind auch Korrekturen leichter durchzuführen.

contents addressable = inhaltsadressierbar
Inhaltsadressierbar sind Speicher, bei denen das Aufsuchen von Speicherplätzen über deren Inhalte erfolgen kann. Besonders nützlich ist das bei Datenbanken, weil es effiziente Suchverfahren ermöglicht. Der Rechner könnte zum Beispiel eine Kundendatei mit Namen, Adressen und Telefonnummern nach der Vorwahl 040 absuchen; dabei käme dann eine Adressenliste der Hamburger Kunden heraus.

Bildnachweise
787: Roy Ingram
788, 808, 809, 811: Ian McKinnell
791, 794, 801: Kevin Jones
793: Liz Heaney
796, 797: Chris Stevens
802: Steve Cross
803: Ford
806: Liz Dixon

Fachwörter von A bis Z

Control Character = Steuerzeichen
Steuerzeichen sind ASCII-Codes, die einen spezifischen Vorgang bei dem Rechner oder einem Peripheriegerät auslösen. Sie gehören nicht zum alphanumerischen Zeichensatz (Buchstaben, Ziffern und Satzzeichen) und sind großenteils nicht ausdruckbar. Viele dieser Symbole sind allgemeingültig festgelegt – das ASCII-Zeichen „13" etwa bedeutet stets Zeilenvorschub – während einige andere je nach Maschinentyp unterschiedlich definiert sind. Die Oric-Rechner arbeiten beispielsweise mit Steuerzeichen zur Verdopplung der Schriftgröße, zur Inaktivierung der Bildschirmdarstellung und zum Farbwechsel. Bei Textverarbeitungssystemen erhält der Drucker über Formatsteuerzeichen Anweisungen für die Textausgabe.

Courseware = Lernsoftware
Der Begriff „Courseware" steht für Lernprogramme, die in der Ausbildung beim computerunterstützten Lernen verwendet werden. Das kann eine Zusammenstellung aus den verschiedensten Quellen und Themengebieten für einen bestimmten Rechner sein, aber auch ein Paket mit einheitlicher Dialogstruktur, wobei die Kursprogramme unterschiedliche Gegenstände behandeln.

CP/M = CP/M
CP/M ist ein Betriebssystem für Rechner auf 8080- oder Z80-Basis, das von Gary Kildall bei Digital Research entwickelt wurde und sich inzwischen als Standard behaupten konnte. CP/M (Control Program for Microprocessors) war das erste universelle Betriebssystem. Es wurde von zahlreichen Herstellern übernommen und ermöglicht die Verwendung einmal erstellter Software auf einer Vielzahl unterschiedlicher Rechner.

Bildnachweise
814, 815, 826, 827: Chris Stevens
816, 817, 819, 834: Liz Dixon
818, 823, 831, 836: Ian McKinnell
821: Rosalind Buckland
825, 830: Steve Cross
829, U3: Kevin Jones
835: Sord

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

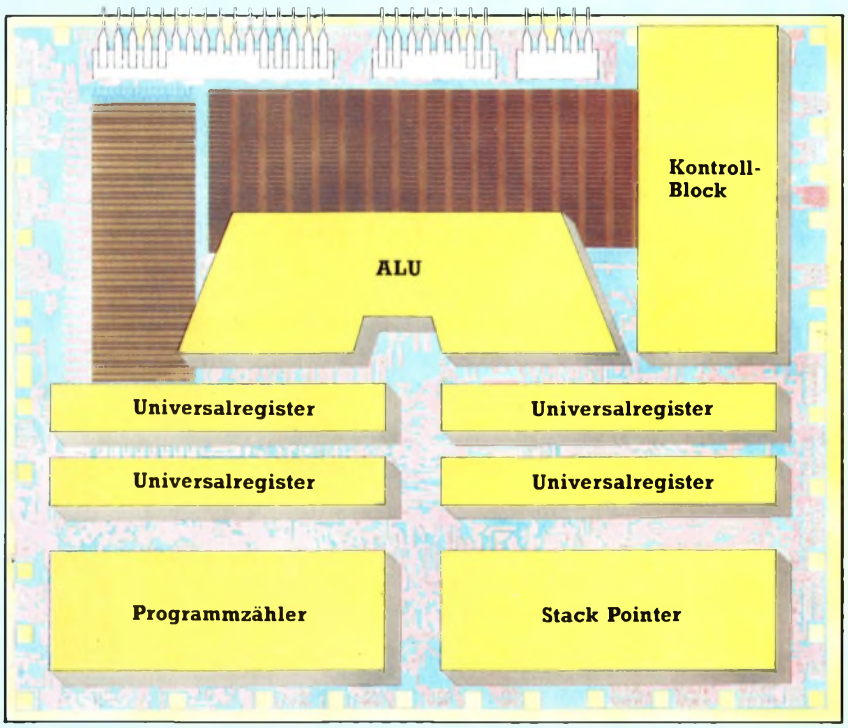
CPU = Zentraleinheit
Die CPU (Central Processing Unit) oder Zentraleinheit eines Computers ist für die interne Organisation aller Arbeitsabläufe verantwortlich.

Die Aufgabe der CPU besteht in der Entgegennahme und Ausführung von Maschinenbefehlen, zum Beispiel für das Umstrukturieren von Speicherinhalten oder für einfache arithmetische Operationen. Alle Zahlen und Befehle werden binär verschlüsselt.

Ein CPU-Baustein enthält viele tausend elektronische Funktionselemente, wobei die Leistungsfähigkeit je nach Typ unterschiedlich ist. Viele Taschenrechner haben eine 4-Bit-CPU, Heimcomputer arbeiten meist mit 8 Bit, eine Reihe von Bürorechnern mit 16 Bit, und mittlere sowie Großrechenanlagen verfügen über eine 32- oder 64-Bit-CPU.

Unterschiede bestehen auch hinsichtlich der in die CPU einbezogenen Funktionen. Der Taktgeber für die unerläßlichen Synchron-Signale ist häufig in die CPU integriert, er kann aber auch als Zusatzbaustein erforderlich sein. Manche CPUs haben intern schon RAMs oder ROMs und Ein/Ausgabe-Baugruppen („Einchip-Microcomputer"), die meisten sind jedoch ohne externe Speicher und andere Bausteine nicht betriebsfähig.

Der Daten- und Steuersignalaustausch mit der CPU erfolgt über drei Leitungssysteme: Daten-, Adreß- und Steuerbus. Diese Leitungen werden an die „Beinchen" (Pins) gelegt, die aus dem Gehäuse der CPU herausragen und mit dem darin verborgenen Siliziumchip über dünne Drähte verbunden sind.



Hier sehen Sie eine typische Microcomputer-CPU mit den Bus-Anschlußstiften. Die überlagerten Blöcke stellen die ALU (Arithmetisch-logische Einheit) mit ihren verschiedenen Komponenten dar.

Fachwörter von A bis Z

Crash = Zusammenbruch
Bei einem Systemzusammenbruch verliert der Rechner sozusagen das Bewußtsein. In diesem Fall hilft nur Drücken des Reset-Knopfs oder Ziehen des Netzsteckers, womit im allgemeinen aber alle Programme und Daten im RAM verloren sind. Deshalb wird der Crash auch oft als „Fatal Error" (vernichtender Fehler) bezeichnet.

Für den Zusammenbruch eines Computersystems gibt es drei Hauptursachen. Häufig liegt die Ursache beim Netzteil. Vor allem die früheren Rechner nahmen schon geringe Spannungsschwankungen äußerst übel. Diese entstehen häufig durch einen lokalen Spannungsabfall bei Anschluß kräftiger Stromverbraucher nahe der Rechner-Steckdose. Bei den neueren Netzteilen ist die Stromversorgung besser geregelt, aber wenn auf Ihren Leitungen „Spikes" (hochfrequente Spannungsspitzen) herumgeistern oder wenn längeranhaltende Spannungseinbrüche auftreten (dabei flackert das Licht), dann sollten Sie sich den Kauf eines externen Netzspannungs-Stabilisators überlegen.

Eine andere Form von Crash wird dadurch ausgelöst, daß der Rechner sich in einer endlosen Programmschleife festläuft. In BASIC hilft dann einfaches Drücken der „Break"- oder „Stop"-Taste, aber Maschinencode-Programme sind nur bei wenigen Rechnern über die Tastatur abzubrechen, so daß eine JMP-Schleife ohne Ausgangsabfrage zwangsläufig zum Crash führt.

Bei den meisten Microprozessoren kommt es außerdem zum Crash, wenn im Befehlsregister ein Opcode auftaucht, der im Befehlsvorrat nicht vorgesehen ist. Dafür kann ein Assemblerfehler verantwortlich sein – vielleicht eine falsche JMP-Anweisung, derzufolge Daten als Befehle interpretiert werden.

Bildnachweise
841, 849, 867: Ian McKinnell
842, 846, 847, 853, 854, 855: Chris Stevens
850, 851, 852, 862: Kevin Jones
856, 857, 866: Liz Dixon
859: Digital Research Inc., Ian McKinnell
860: Digital Research Inc.
864: Simon Daton
865: Steve Cross, Ian McKinnell, Dimension Graphics

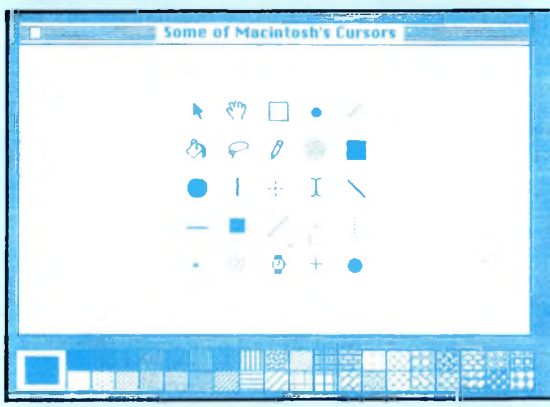
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Cross-Assembler = Cross-Assembler
Ein Cross-Assembler ist ein Übersetzungsprogramm zur Umwandlung vom Source Code in den Object Code. Cross-Assembler erlauben den Entwurf von Programmen für unterschiedliche Microcomputer auf ein und demselben Entwicklungssystem, das mit viel mehr Komfort und Speicherplatz ausgestattet sein kann als ein Heimcomputer, natürlich auch schneller ist und über eine größere Auswahl an Editier- und Dienstprogrammen verfügt. Der Einsatz ist dann sinnvoll, wenn Sie zum Beispiel Ihren Lebensunterhalt damit verdienen wollen, Spiel- oder Geschäftsprogramme zu entwerfen und selbständig zu vertreiben.

Das Entwicklungssystem kann sogar einen anderen Prozessor haben als der Rechner, für den die Programme entwickelt werden. Der Cross-Assembler wird mit den unterschiedlichsten Befehlsstrukturen und auch mit Adressierungsproblemen fertig. Ein Cross-Compiler arbeitet ähnlich, dient jedoch zur Übersetzung von Programmen, die in einer Hochsprache und nicht in Assembler geschrieben sind.

Current Loop = Stromschleife
Die gängigste Norm für den seriellen Datenverkehr ist der RS232-Standard, der die Codierung der Datenbits (mit Start- und Stop-Bits) und das „Handshaking" für die Steuerung der Übertragung festlegt.

Der US-Norm RS232 entspricht die internationale CCITT-Norm V.24. Dabei werden die Datenbits durch einen Spannungspegel von -3 bis -15 V für die logische Eins und einen Pegel von +3 bis +15 V für die logische Null dargestellt. Neben dieser „Spannungsschnittstelle" gibt es noch die 20 mA-Stromschnittstelle (TTY), die ursprünglich zur Ansteuerung von Fernschreibern verwendet wurde. Dabei bilden die Peripheriegeräte mit Hin- und Rückleitung eine geschlossene Stromschleife (Current Loop), in der eine 1 durch einen Strom von 20 mA und eine 0 durch eine Stromstärke von Null Milliampere realisiert wird.



Cursor = Cursor
Der Cursor ist ein Bildschirmsymbol, das die nächste Zeichenposition andeutet. Das Wort (ursprünglich aus dem Lateinischen: cursor = der Läufer) bezeichnet im Englischen auch den Schieber des Rechenstabs oder den Stellenanzeiger bei mechanischen Addiermaschinen.

Traditionell wird als Cursor bei Computern ein blinkendes Unterstreichungszeichen oder ein Quadrat verwendet, manchmal auch einfach ein blinkender Buchstabe. Dagegen setzen Rechner wie der Lisa von Apple oder der Macintosh als Cursor nach Bedarf verschiedene Piktogramme ein: Eine Armbanduhr zeigt Ihnen beispielsweise, daß Sie warten müssen, und eine Hand deutet darauf hin, daß eine Objektbewegung erwartet wird.

Fachwörter von A bis Z

Daisy Wheel = Typenrad
Für Computerbesitzer, die einen Schönschreibdrucker brauchen, kommt fast nur das Typenradsystem in Frage. Der Name rührt von der Anordnung der Drucktypen her: Sie sitzen auf den Kunststoffspeichen eines Rades (Wheel) von circa 7 cm Durchmesser wie die Blütenblätter eines Gänseblümchens (Daisy). Das Rad dreht sich, bis der gewünschte Buchstabe oben steht, dann schlägt die Type auf Farbband und Papier, wodurch das entsprechende Schriftbild erzeugt wird.

Typenraddrucker sind erheblich langsamer als die Matrixdrucker. Ein 1500-Mark-Typenraddrucker schafft nur 10-15 Zeichen/s, ein Matrixdrukker dagegen glatt das Zehnfache. Auch die bis zu 8000 Mark teuren Typenrad-Spitzenmodelle erreichen nicht mehr als 120 Zeichen/s, weil das Typenrad vor jedem Anschlag durch den Antriebsmotor einmal beschleunigt und wieder abgebremst werden muß. Bei näherer Betrachtung des Typenrads erscheint die Reihenfolge der Buchstaben zunächst willkürlich – sie ist aber so gewählt, daß unter Berücksichtigung der üblichen Buchstabenkombinationen möglichst kleine Drehwinkel erforderlich sind.

Database = Datenbank
Ein Datenbanksystem ist ein mehr oder weniger universelles Softwarepaket zum Abspeichern und Lesen von Informationen. Die Daten über eine Person (in einer Adreßliste) oder einen Artikel (bei der Lagerverwaltung) sind jeweils in einem Datensatz (record) enthalten, der einer Karteikarte bei konventionellen Karteien entspricht. Jeder Datensatz besteht aus einzelnen namentlich gekennzeichneten Feldern.

Bei einfachen Datenbanksystemen läßt sich nicht mehr als eine Datei auf einmal bearbeiten, während bei Mehrdatei-Systemen ein Datenaustausch zwischen verschiedenen Dateien möglich ist. Ein Abrechnungsprogramm kann auf diese Weise Daten aus einer Kundendatei, in der Kundenadressen und Höchstkredite

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

vermerkt sind, mit Preisen, Artikelnummern und detaillierten Beschreibungen aus einer Produktdatei zusammenbringen. Ausgefeilte Datenbanksysteme sind programmierbar: Die Software „erlernt" gängige Befehlssequenzen und führt sie dann auf einen einzigen Tastendruck hin aus. Solche Systeme werden auch als „Anwendungsgenerator" bezeichnet.

Der eigentliche Sinn eines Datenbanksystems besteht darin, daß man Datensätze nach verschiedenen Gesichtspunkten sortieren kann. Zur Veranschaulichung ist hier der bekannte „Zauberwürfel" von Rubik gezeigt, der die Datenbank darstellen soll; die einzelnen Facetten der Würfelseiten entsprechen den noch ungeordneten Datensätzen.

Data Compression = Datenverdichtung
Im allgemeinen sieht man als Speicherbedarf je Zeichen ein Byte (= 8 Bit) vor und zusätzlich etwas Platz für die Dateiorganisation in Form von Kopfsätzen und Steuerzeichen. Durch „Datenverdichtung" oder „Komprimierung" können Sie den Bedarf jedoch reduzieren. Sie gewinnen damit einerseits Speicherkapazität und verringern andererseits die Datenübertragungszeit.

Jede Verdichtung erscheint zunächst reichlich kompliziert, zumal alle Zeichen und viele Befehle ASCII-verschlüsselt sind. Aber denken Sie an einen Text, der nur aus Kleinbuchstaben und maximal sechs Satzzeichen besteht – da reichen fünf Bits, um die erforderlichen 32 (= 2^5) Symbole darzustellen. Statt für jedes Zeichen ein ganzes Byte zu opfern, lassen sich die Daten vor dem Abspeichern auf Diskette durch ein Spezialprogramm komprimieren und nach dem Lesen durch ein anderes Programm wieder auf die normale Länge bringen. Als Nebeneffekt bieten komprimierte Dateien einen gewissen Schutz gegen Mißbrauch, weil sie nicht ganz so leicht von einem fremden System gelesen werden können.

Ein stärkeres Verdichten als im Verhältnis 8:5 ist mit Hilfe von Häufigkeitsanalysen möglich. Bestimmte Buchstabenfolgen sind besonders häufig – ein Durchschnittstext besteht etwa zur Hälfte aus rund hundert Wörtern wie „und", „der/die/das" oder „ist". Stellt man diese Wörter durch je ein Byte (mit einem Kennbit zur Unterscheidung von Einzelzeichen) dar, ergibt sich eine beträchtliche Platzeinsparung.

Bildnachweise
870: Associated Press
871: Associated Press, Ian McKinnell
873, 877, 881, 885, 889, 895: Ian McKinnell
874, 879, 884: Liz Dixon
875, 882: Chris Stevens
878: Alison Fenton
883, 891, 896: Kevin Jones
888: Ian McKinnell, Liz Dixon
894: Steve Cross

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Data Processing = Datenverarbeitung
Früher hieß alles „elektronische Datenverarbeitung", was mit Computern zu tun hatte. Vor dem Einzug der Microprozessoren gab es neben Großrechnern nur sehr teure kleinere Anlagen. Diese waren für Privatleute ungeeignet, unter anderem deshalb, weil sie ein bestimmtes Raumklima und ständigen Service benötigten. Die Großrechner wurden durch geschultes Personal bedient, und ohne Genehmigung hatte niemand Zugang zur Maschine.

Die Microcomputer-Revolution fegte die ganze Exklusivität hinweg. Der Micro stellt keine besonderen Ansprüche an Aufstellungsort und Bedienung und kann als „Arbeitsplatzrechner" auf dem Schreibtisch jedes Angestellten ständig für alle möglichen Zwecke bereitstehen. In großen Firmen gibt es daneben weiterhin ein richtiges Rechenzentrum, das für übergreifende Aufgaben wie Finanzplanung, Lagerverwaltung, Buchführung, Produktionsabwicklung und den Personalbereich mit Lohn- und Gehaltsabrechnung zuständig ist.

Debugging = Fehlerbeseitigung
Debugging ist die Beseitigung von Störungen im weitesten Sinn, seien es nun Programmfehler oder Defekte in der Hardware. Im engeren Sinn wird das Wort für das systematische Testen und Korrigieren von Software verwendet.

Der Begriff ist von Grace Hopper geprägt worden, die 1945 bei der Entwicklung des Relais-Rechners „Harvard Mark II" mitarbeitete. Als dieser plötzlich seine Dienste verweigerte, suchte man lange nach der Ursache – diese zeigte sich dann auch, in Form eines Nachtfalters (bug), der sich zwischen den Relais verklemmt hatte. Seither ist das Wort „Bug" für Fehler jeglicher Art in Gebrauch, auch für Softwarefehler.

Decision Table = Entscheidungstabelle
Die Flexibilität eines Rechners hängt wesentlich davon ab, wie Entscheidungen realisiert werden können. Das einfachste Beispiel ist die IF... THEN – Anweisung in BASIC, die ein bestimmtes Vorgehen veranlaßt, wenn die definierten Bedingungen erfüllt sind.

In einer Entscheidungstabelle sind die Bedingungen und die möglichen Verfahrensweisen systematisch zusammengestellt.

In BASIC ist der Aufbau einer Entscheidungstabelle, außer in Form einer Serie von IF... THEN-Anweisungen, eine komplizierte Sache. Leichter realisierbar ist die Tabelle mit ON... GOTO-Anweisungen, wobei die Bedingungen durch Zahlen gekennzeichnet werden und die zugehörigen Vorgehensweisen über Sprungbefehle angesteuert werden.

Decision Tree = Entscheidungsbaum
Bei einem Entscheidungsbaum liegen Bedingungen und zugeordnete Vorgehensweisen in anderer Struktur vor als bei der Entscheidungstabelle. Der Rechner geht nicht schrittweise die ganze Bedingungsliste durch, sondern wird von einer Eingangsliste über Verzweigungen an immer detailliertere Teillisten weiterdirigiert.

Declaration Statement = Vereinbarung
In BASIC müssen alle Variablenfelder vor Gebrauch DIMensioniert werden. Einige andere Sprachen verlangen eine ähnliche Spezifizierung für sämtliche Variablen. Der Benutzer muß definieren, ob es Integer- oder Real-, Double-Precision- oder String-Variable sein sollen. Alle Anweisungen dieser Art werden als Vereinbarungen bezeichnet.

Die Vereinbarung von Feldern dient eigentlich nur zur Verbesserung der Leistungsfähigkeit. DIM-Anweisungen veranlassen den BASIC-Interpreter, im RAM einen Bereich von der Größe der Felder zu reservieren. Gewöhnliche Variable werden im Anschluß daran gespeichert. Ohne vorherige Dimensionierung müßte das Betriebssystem beim Auftreten eines neuen, höheren Feldindex jedesmal alle folgenden Variablen im RAM weiterschieben, um Platz zu schaffen.

Auch in BASIC ist es zweckmäßig, eine Vereinbarung aller Variablen zu simulieren, indem man sie in den ersten Programmzeilen in der Reihenfolge der Häufigkeit ihrer Benutzung einmal erscheinen läßt. So können die notwendige Rechenzeit und der benötigte Speicherplatz wesentlich verringert werden.

Bildnachweise
897: Liz Heaney, Pilot Software
903, 908, 915, 918, 919, 924: Ian McKinnell
905, 912, 913, 914: Liz Dixon
907: Ian McKinnell, Microscope
909, 910, 911: Chris Stevens
916: Kevin Jones
917: Ian McKinnell, Dave Whelan

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Decrement = Dekrementieren
Dekrementieren heißt Vermindern. In der Datenverarbeitung ist dabei die Verkleinerung eines Zählerinhalts gemeint. In BASIC wird zum Beispiel durch die Anweisung LET A = A- 1 der Index A dekrementiert. Wichtigster Anwendungsbereich sind Schleifen, wobei mit jedem Durchlauf der Schleifenzähler um 1 dekrementiert wird. Bei den meisten BASIC-Versionen sind dafür mehrere Anweisungsformen mit automatischer Indexfortschaltung und Endabfrage vorgesehen, etwa FOR ... NEXT, REPEAT ... UNTIL oder WHILE ... END.

Im Maschinencode gibt es für das Dekrementieren spezielle Befehle, etwa das Kommando DEC, das ein im Adreßteil angegebenes Byte dekrementiert. DEX und DEY können zum Beispiel die Indexregister X und Y dekrementieren. Eine andere wichtige Anwendung ist die indizierte Adressierung, wenn eine ganze Serie von Speicherbytes nach der gleichen Vorschrift verarbeitet werden soll. Das Gegenteil von Dekrementieren, nämlich die schrittweise Erhöhung eines Indexregisters, heißt Inkrementieren.

Degaussing = Entmagnetisierung
Nach längerem Gebrauch wird der Schreib/Lese-Kopf eines Cassettenrecorders allmählich permanentmagnetisch. Dieser Effekt führt zu Signalverzerrungen, die außerordentlich störend werden können. HiFi-Fans begegnen dem durch regelmäßiges Entmagnetisieren.

Wer als Heimcomputer-Benutzer auf den Cassettenrecorder angewiesen ist, sollte die Wartung nicht vernachlässigen. Magnetköpfe und Laufwerk müssen regelmäßig gereinigt werden, um den normalen Bandabrieb zu entfernen, und alle paar Monate sollten auch hier die Köpfe entmagnetisiert werden. Dazu brauchen Sie ein Gerät, das ähnlich aussieht wie eine Taschenlampe. Es enthält eine Spule, deren Eisenkern einseitig aus dem Gehäuse hervorragt. Der Apparat wird ans Netz angeschlossen, an den Magnetkopf gehalten und vor dem Abschalten langsam wieder vom Recorder entfernt. Dadurch wird der Kopf einem magnetischen 50-Hz-Wechselfeld mit abnehmender Amplitude ausgesetzt, womit die Dauermagnetisierung auf einfachste Weise gelöscht wird.

Delimiter = Trennzeichen
Trennzeichen markieren Anfang und Ende von Datenblöcken im RAM, können aber auch Datensätze bei Diskettendateien abgrenzen.

Der Gebrauch von Trennzeichen läßt sich anhand der Speicherung eines BASIC-Programms erläutern. In den beiden ersten Bytes jeder Zeile steht die Zeilennummer als 16-Bit-Integer. Die beiden nächsten Bytes enthalten die RAM-Adresse, unter der die nächste Zeile steht – das beschleunigt das Aufsuchen von Zeilennummern bei GOTO- oder GOSUB-Anweisungen. Danach steht der Zeilentext, gefolgt von einem Trennzeichen, meist in Form eines Nullbyte, zur Markierung des Zeilenendes. Das Trennzeichen darf dabei keinesfalls als Teil der Information zu mißverstehen sein.

Diagnostic Routine = Diagnoseprogramm
Die Vorstellung, daß ein Rechner Störungen von sich aus mitteilt, hat einigen Reiz, obwohl damit vielen Zukunftsromanen der Boden entzogen wäre. So etwas gibt es tatsächlich schon in Form von Diagnoseprogrammen bei einigen Großrechenanlagen und (leider nur wenigen) Microcomputern.

Kein System kann alle Fehler anzeigen. Wenn etwa das Netzteil ausfällt, rührt sich auch das Diagnoseprogramm nicht mehr. Dagegen kann es durchaus die Funktion von Speicher- und Schnittstellenbausteinen oder ganzen Logikplatinen testen. Bei den meisten Micros ist eine einfache RAM-Prüfung vorgesehen, bei der jedes Byte einmal beschrieben und wieder gelesen wird. Dieses Verfahren dient nicht nur zum Erkennen von Speicherfehlern.

Eines der besten Diagnose-Systeme im Microcomputer-Bereich bietet der „Rainbow" von Digital Equipment. Jede Platine kann softwaremäßig vollständig getestet werden, und bei Fehlermeldungen wird die Lage des defekten Teils auf dem Bildschirm angezeigt.

Beim NewBrain-Rechner erscheinen auf dem einzeiligen Display geheimnisvolle Hieroglyphen, während das Diagnoseprogramm läuft.

Digital = Digital
Bei einem digitalen Signal kann der Spannungspegel nur „diskrete", das heißt, abgestufte, vorgegebene Werte annehmen – bei Digitalrechnern genau zwei. Im Gegensatz dazu ist ein Analogsignal „stetig" veränderbar und kann beliebige Zwischenwerte annehmen.

Bildnachweise
925: Steve Cross
927: Liz Heaney
928, 946, 947: Liz Dixon
932, 944, 945, 952, U3: Ian McKinnell
933, 936, 949: Kevin Jones
937, 938, 939: Chris Stevens
950: Bob Bromide, Tony Sleep
951: Tony Sleep

Fachwörter von A bis Z

Digital Plotter = Digitalplotter
Bei den Zeichengeräten für Heimcomputer handelt es sich fast durchweg um Digitalgeräte. Sie werden über die Druckerschnittstelle betrieben und interpretieren die empfangenen Daten als x,y-Koordinaten von Punkten, die zu Papier gebracht werden sollen. Früher gab es nur Analogplotter, vorwiegend für den Einsatz in der Meßtechnik, etwa um die Temperatur im Innern eines Schmelzofens im genauen Zeitablauf zu zeichnen.

Digitise = Digitalisieren
Digitalisieren ist das Umsetzen einer analogen Größe in einen entsprechenden Digitalwert. Bei der Sprachdigitalisierung wird das Ausgangssignal eines Mikrofons einem Analog/Digital-Wandler zugeführt und digital auf Diskette gespeichert. Dieses Verfahren wird allerdings meist nicht als Digitalisierung, sondern als „Abtastung" (Sampling) bezeichnet, weil das Analogsignal periodisch abgefragt und gewandelt wird. Wie sich mathematisch beweisen läßt, muß zur getreuen Wiedergabe eines Signals die Abtastrate mindestens doppelt so hoch sein wie die höchste im Signal enthaltene Sinusfrequenz. Wenn beispielsweise wie beim Telefon ein Frequenzbereich von 300–3400 Hz erfaßt werden soll, muß die Abtastrate 6,8 kHz betragen.

Digitiser = Digitalisiergerät
Bei vielen Rechneranwendungen, insbesondere bei der Konstruktion, Kartographie und Entwurfsgestaltung sind Zeichnungen vom Papier auf den Computer zu übertragen. Dazu brauchen Sie ein Digitalisiergerät mit schnellem Analog/Digital-Wandler. Es genügt aber in vielen Fällen auch ein Digitalisiertablett, bei dem die Zeichnung von Hand mit einem speziellen Stift nachgefahren wird.

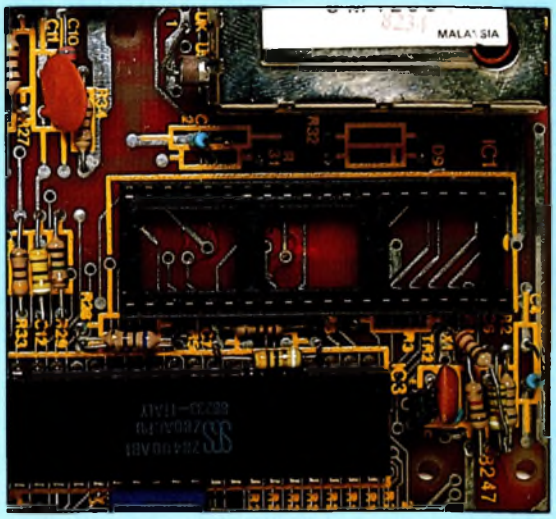
Zur Ermittlung der Stiftposition gibt es unterschiedliche Verfahren. Bei einigen Geräten ist in das Tablett ein Drahtgitter eingebettet, über das der Stift „kapazitiv" oder „induktiv" geortet wird. Andere Systeme arbeiten mit Ultraschall, indem sie mit zwei Detektoren unter verschiedenen Winkeln die Entfernung zur Stiftspitze messen.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

DIL = DIL
Dual In-Line kennzeichnet die übliche Gehäusebauform für integrierte Schaltungen mit „zweireihiger" Anordnung der Steckerstifte. Die Chips werden heute meist direkt eingelötet. So können die Herstellkosten gesenkt werden. Die Leiterplatten-Unterseite wird dazu in flüssiges Zinn getaucht, nachdem die Pins zuvor mit Bestückungsmaschinen in die entsprechenden Bohrungen der Platine gesteckt wurden.

Früher wurden nicht die Chips selbst eingelötet, sondern in DIL-Sockel gesetzt, die danach festgelötet wurden. Das erhöhte zwar die Fertigungskosten, defekte Chips ließen sich aber schnell ohne zu löten austauschen. Die Bausteine sind jedoch in den letzten zehn Jahren so viel zuverlässiger geworden, daß häufig der gelegentliche Ersatz einer ganzen Karte für den Hersteller billiger ist als die Verwendung von Sockeln und der Austausch einzelner Chips.

Bildnachweise
953, 954, 972, 973: Chris Stevens
958, 968, 969: Kevin Jones
962, 963, 964, 970, 971: Liz Dixon
964, U3: Ian McKinnell
974: Your Spectrum



Das Foto zeigt eine Heimcomputer-Platine mit einem gängigen 40poligen DIL-Sockel oberhalb des Prozessorchips.

Dimension = Dimension
Die Anzahl der Dimensionen eines Systems gibt an, wieviele Parameter zur eindeutigen Definition eines Systemelements erforderlich sind. Allgemein geläufig sind dreidimensionale Systeme, bei denen drei Achsenkoordinaten einen Raumpunkt bestimmen.

In der Datenverarbeitung spielen Dimensionen bei Variablenfeldern eine wichtige Rolle. Das Feld A(5,6) beispielsweise ist zweidimensional, weil zwei Indizes zur Festlegung eines Elements nötig sind.

Direct Access = Direktzugriff
Ein direkter Zugriff auf Daten ist (insbesondere bei Diskettendateien) dann gegeben, wenn jede Informationseinheit unmittelbar angesprochen werden kann. Bei seriellem Zugriff müssen Sie dagegen jedesmal eine Menge nicht benötigter Daten durchgehen. Wenn eine Datei (wie bei einer Datenbank) aus einzelnen Datensätzen aufgebaut ist, muß das Betriebssystem für den Direktzugriff eine ständig aktualisierte Liste führen, in der die Anfangsadressen aller Datensätze der Datei stehen. Zum Abruf einer speziellen Information steuert das DOS anhand dieser Tabelle den Lesekopf unmittelbar auf die entsprechende Spur und sucht den entsprechenden Sektor auf.

Fachwörter von A bis Z

Disassembler = Disassembler
Ein Disassembler ist ein Programm, das Maschinencode in Assemblersprache zurückübersetzt. Es wandelt die Bytes in mnemotechnische Abkürzungen wie LDA oder JMP um, analysiert die Adressierungsart und gibt auch den Operanden mit aus, der in den nächsten Bytes steht.

Disassembler sind sehr nützlich, wenn Maschinenprogramme zu testen oder umzuschreiben sind.

DMA = DMA
DMA heißt „Direct Memory Access", auf deutsch „direkter Speicherzugriff". Darunter versteht man schaltungstechnische Maßnahmen, die mehreren Systemkomponenten gleichzeitig den Zugang zum selben Speicher erlauben. Eine wichtige Anwendung ergibt sich bei der Grafikprogrammierung: Wenn der Video-Steuerchip den Bildspeicher selbst auslesen kann, ohne für jedes Byte die CPU zu befragen, wird die Sache wesentlich effizienter.

Double Density =
Doppelte Aufzeichnungsdichte
Die Kapazität einer Diskette hängt einmal davon ab, ob sie einseitig (Single Sided = SS) oder doppelseitig (Double Sided = DS) beschichtet und damit bespielbar ist, außerdem aber von der Aufzeichnungsdichte (Spurendichte und Bitdichte in der Spur). Bei Disketten mit doppelter Dichte (Double Density = DD) ist auf einer Seite die zweifache Datenmenge unterzubringen wie bei einfacher Dichte (Single Density = SD). Die DD-Aufzeichnung erfordert eine wesentlich präzisere Magnetkopfsteuerung, was anfänglich mit merklich höheren Kosten und geringerer Zuverlässigkeit verbunden war.

Infolge des Fortschritts der Diskettentechnologie in den letzten Jahren ist doppelte Dichte inzwischen praktisch Standard. Auf einer 5 1/4-Zoll-Diskette sind je nach Formatierung und System 90 KByte bis 1 Megabyte an Daten unterzubringen und auf den neueren 3 1/2-Zoll-Disketten bis zu 700 KByte.

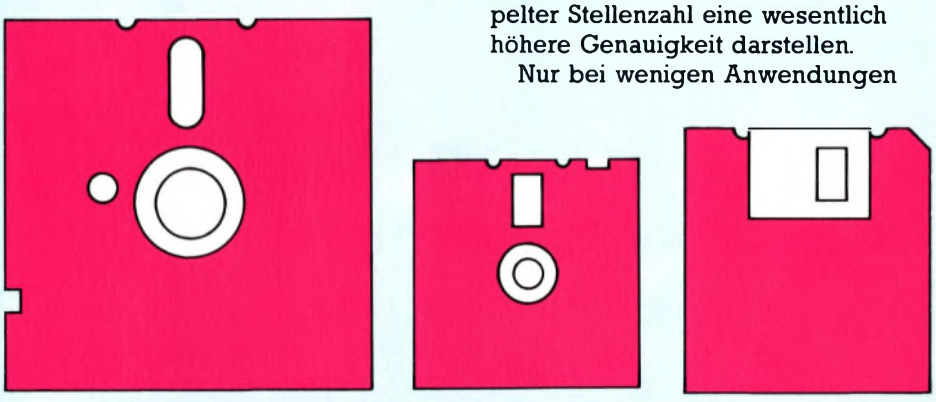
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Double Precision =
Doppelte Genauigkeit
Reale Zahlenvariablen, wie sie in BASIC am häufigsten sind, werden im allgemeinen mit acht oder neun Dezimalstellen gespeichert. Dabei spricht man von „Single Precision" (einfacher Genauigkeit), wogegen Double-Precision-Variablen mit doppelter Stellenzahl eine wesentlich höhere Genauigkeit darstellen.

Nur bei wenigen Anwendungen benötigt man Ergebnisse, die mehr als acht Stellen haben – wozu also 16 Stellen? Deren Notwendigkeit ergibt sich aus den Rundungsfehlern, die bei jeder Rechenoperation wie Addieren, Subtrahieren usw. entstehen. Bei umfangreichen Berechnungen, die beispielsweise beim Maschinenbau, bei Statistiken und Wetterprognosen durchgeführt werden, muß der Programmierer genau darauf achten, daß keine Rundungsfehler entstehen, die das Ergebnis unkontrollierbar verfälschen. Das Rechnen mit doppelter Genauigkeit entschärft dieses Problem, aber ohne es aus der Welt zu schaffen.

Download =
Laden (aus Fremdsystemen)
Unter „Telesoftware" versteht man Programme, die aus einer Zentrale über Telekommunikation abrufbar sind, und als Downloading werden die Übernahme und das Abspeichern im RAM oder auf Diskette bezeichnet. Ursprünglich wurde dieser Begriff bei der Übertragung von Daten aus einem Großrechner zu den angeschlossenen Terminals oder kleineren Rechnern verwendet. In England können Sie über das PRESTEL-Bildschirmtextsystem bereits telefonisch (gegen Gebühr) viele Programme für Ihren Heimcomputer abrufen.

Das Downloading kann auch den Softwareverkauf in Zukunft ganz anders gestalten. Zur Zeit ist die Lagerhaltung für die Händler ein echtes Problem, weil sich die Popularität von Spielprogrammen ständig ändert. Ein Lösungsversuch sind nachprogrammierbare Cartridges, die der Benutzer im Geschäft preisgünstig neu laden lassen kann. Ein spezielles Terminal überträgt dabei das Programm von einer Zentrale auf die Cartridge.



Die Spurendichte und die Bitdichte innerhalb einer Spur sind vom Diskettendurchmesser völlig unabhängig. Abgebildet sind hier im gleichen Maßstab (1) die 5 1/4-Zoll-Minifloppy, (2) die 3-Zoll- und (3) die 3 1/2-Zoll-Microfloppy.

Bildnachweise
981, 982, 983, 992, 996, 998, 1003: Ian McKinnell
984, 985, 990: Kevin Jones
989: Liz Heaney, Kevin Jones
993, 994: Chris Stevens
1001, 1007, 1008, U3: Liz Dixon

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Drop-In = Störsignal
Von einem Drop-In spricht man, wenn auf einem Magnetträger ein Zeichen erscheint, wo keines hingehört. Die Ursache für Drop-Ins sind meist defekte Stellen in der Magnetschicht. Genau wie Drop-Outs bringen auch einzelne Drop-Ins nicht umbedingt den Datenverkehr durcheinander, weil die meisten Betriebssysteme Lesekontrollen beinhalten, die gelegentliche Bitfehler identifizieren und zum Teil auch korrigieren können.

Drop-Out = Signalausfall
Ein Drop-Out tritt auf, wenn sich aus der Beschichtung einer Diskette oder eines Magnetbands winzige Partikel lösen – Billigcassetten sind dafür besonders anfällig. Durch Drop-Outs kann ein gespeichertes Programm völlig unbrauchbar werden. Deshalb sollten Magnetträger stets mit besonderer Sorgfalt behandelt werden.

Disketten und Cassetten, die zu Drop-Outs neigen, werfen Sie am besten weg. Aber was machen Sie bei einer Winchesterplatte, die ein paar tausend Mark gekostet hat? Die Frage löst für Sie das DOS, das bei Festplatten wesentlich raffinierter arbeitet als bei Disketten: Wenn durch Fehler beim Prüflesen ein „Bad Block" erkannt wird, merkt sich das DOS dessen Lage und sperrt ihn für die weitere Benutzung.

Dump = Speicherabzug
Mit einem Dump wird der Inhalt eines Speichers auf dem Bildschirm oder durch Ausdrucken auf Papier sichtbar gemacht. Ein Binary Dump liefert die Information Byte für Byte binär verschlüsselt, ein Hex Dump hexadezimal usw. Ein Dump ist nicht nur nützlich, wenn Sie ein Maschinencode-Protokoll für die Eingabe in Steuerungssoftware brauchen, sondern hilft auch bei der Fehlersuche.

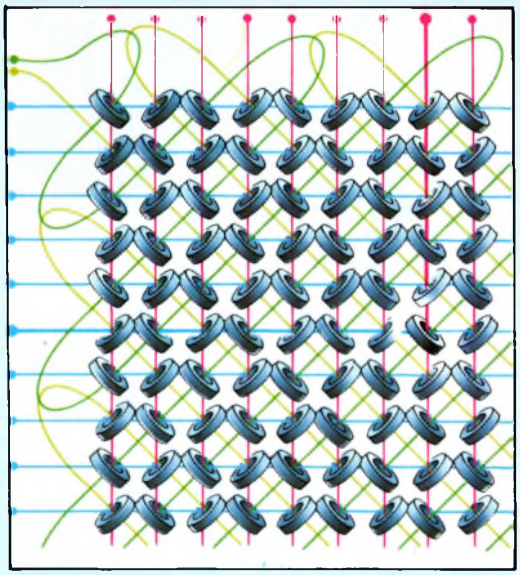
Duplex = Duplex
Beim Duplex-Betrieb können auf einem Kanal gleichzeitig in beiden Richtungen Informationen übertragen werden. Von Halbduplex spricht man, wenn der Hin- und Rückverkehr nur im Wechsel von Senden und Empfangen möglich ist. Viele Gegensprechanlagen und Funkgeräte arbeiten im Halbduplex-Betrieb, wobei jeweils von Sprechen auf Hören umgeschaltet wird. Auch der Datenverkehr zwischen Computern auf Telefonleitungen erfolgte früher halbduplex, und einen Voll-/Halbduplex-Umschalter haben auch heute noch einige Modems – der Vollduplex-Betrieb erfordert eine größere Übertragungs-Bandbreite.

Gleichzeitiges Senden und Empfangen von Informationen wird selten praktiziert. Der Vollduplexbetrieb erlaubt in Form einer simultanen Rückmeldung über den Datenempfang aber eine wirksame Übertragungskontrolle. Werden Fehler festgestellt, etwa infolge eingekoppelter Störungen, kann der Empfänger die entsprechenden Zeichen noch einmal anfordern.

Dynamic RAM = Dynamisches RAM
Bei Microcomputern werden statische und dynamische RAMs eingesetzt. Beide sind flüchtig, der Inhalt geht also bei Stromabschaltung verloren. Ein statisches RAM besteht aus zahllosen Flipflops (eins für jedes Speicherbit) und benötigt für den Verkehr mit dem Prozessor keine externe Beschaltung.

Dynamische RAMs erfordern mehr Aufwand, sind aber schneller, billiger und bieten in einem Baustein mehr Kapazität. Für jedes Bit ist ein kleiner Kondensator (mit nachgeschaltetem Transistor) vorgesehen, der je nach Ladezustand eine 1 oder eine 0 darstellen kann. Da die Ladung abfließt, muß das RAM durch eine spezielle Schaltung alle paar Millisekunden „aufgefrischt" werden. Die „Refresh"-Schaltkreise werden neuerdings gleich in die RAM-Bausteine integriert. Das erleichtert den Systemaufbau erheblich.

Der Kollektorstrom eines Transistors mit geöffnetem Gehäuse ändert sich entsprechend der Belichtung. Auch ältere Ausführungen dynamischer RAMs zeigen eine solche Lichtempfindlichkeit und eignen sich daher zum Aufbau von preiswerten visuellen Sensoren für Roboter.



Ferritkernspeicher
RAMs sind heute durchweg Halbleiterspeicher, aber lange Zeit waren die hier gezeigten Kernspeicher mit kleinen Ferritringen die preisgünstigsten und schnellsten Speicher für direkten Zugriff. 1 und 0 werden durch entgegengerichtete Magnetisierung der Kerne definiert, die auf Schreib- und Lesedrähte aufgefädelt sind.

Bildnachweise
1009: Steve Cross
1010, 1011, 1012: Kevin Jones
1015, 1020, 1024, 1025, 1028, 1029, 1032: Liz Dixon
1016: Ian McKinnell, Quicksilva, Kevin Jones, Your Spectrum
1017, 1018, 1021, 1022, 1023, 1035, 1036: Chris Stevens
1019, 1027, 1031: Ian McKinnell

Fachwörter von A bis Z

EAROM = EAROM
Der elektrisch veränderbare Festwertspeicher (Electrically Alterable Read-Only Memory) gehört zu den neueren Halbleiterspeichern. Seine Funktion ist am besten anhand einer Gegenüberstellung der ROM-Typen zu verstehen. Ein gewöhnliches ROM ist „maskenprogrammiert", das heißt, sein Inhalt wird beim Herstellungsprozeß durch die Belichtungsmaske festgelegt. Ein PROM (Programmierbares ROM) ist ein leeres ROM, das vom Benutzer mit einem PROM-Brenner beschrieben, aber nicht mehr geändert werden kann. Beim EPROM (Erasable = löschbares PROM) ist über dem Siliziumchip ein Quarzfenster angebracht, durch das sich der Speicher mit UV-Licht löschen und dann neu programmieren läßt.

Ein EEROM (Electrically Erasable = elektrisch löschbares ROM) funktioniert ähnlich wie ein EPROM, wird aber elektrisch und nicht mit UV-Licht gelöscht. Im Gegensatz zum EEROM können bei einem EAROM die Speicherplätze einzeln gelöscht bzw. überschrieben werden, was allerdings einigen Aufwand voraussetzt und wesentlich langsamer geht als das Lesen. Deshalb werden EAROMs vor allem dann verwendet, wenn Speicherinhalte geringfügig verändert werden müssen.

Edge Connector = Platinenstecker
Der Platinenstecker ist die einfachste Form des Rechneranschlusses und findet sich hauptsächlich bei Heimcomputern der unteren Preisklasse. Die Kontaktbahnen werden bei der Platinenherstellung im normalen Ätzverfahren auf einem etwa 1 cm überstehenden Randstreifen angelegt. Hochwertige Stecker haben vergoldete Kontakte, denn Kupfer oxidiert an der Luft, was Übergangs-

Bildnachweise
1037, 1038, 1051, 1054: Chris Stevens
1041: David Higham
1042: Liz Heaney
1043, 1044, 1045, 1050, 1053, 1060, 1061, 1063, 1064: Ian McKinnell
1058, 1059: Liz Dixon

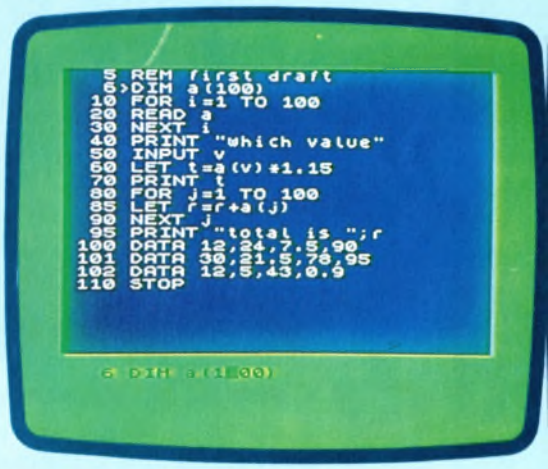
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

schwierigkeiten verursachen kann. Bei teureren Rechnern werden statt Platinensteckern spezielle Interfacebuchsen verwendet.

Editor = Editor
Ein Editor ist ein Systemprogramm, das den Benutzer bei der Erzeugung und Abänderung von Zeichenfolgen unterstützt. Dabei kann es sich um Grafik-Symbole, BASIC-Anweisungen oder (bei der Textverarbeitung) um Sätze handeln. Zum Editieren benötigt man eine Reihe elementarer Funktionen wie Einfügen und Löschen von Zeichen, Überschreiben usw. Hochwertige Systeme bieten jede dieser Funktionen in verschiedenen Varianten – der Benutzer kann zum Beispiel Einzelbuchstaben, Wörter, Sätze, Abschnitte oder ganze Seiten löschen. Beim „bildschirmorientierten" Editor können Sie die Änderungen an jeder Stelle des Schirms durchführen, beim „zeilenorientierten" dagegen nur in der vorher definierten Zeile.

Electronic Mail = Elektronischer Briefkasten
Die Bürokommunikation entwickelt sich zu einem immer bedeutenderen Einsatzfeld für Microcomputer – zunehmend wird von der Datenübermittlung über Fernsprechleitungen Gebrauch gemacht. Ein verbreitetes Standard-System ist die elektronische Post, die den Austausch von Schriftstücken und Nachrichten über Rechner ermöglicht. Jeder Teilnehmer braucht neben dem Heimcomputer ein Modem und spezielle Software. Kurze Meldungen können über die Rechnertastatur direkt auf die Leitung gehen. Längere Schreiben speichert man üblicherweise auf Diskette oder Cassette ab und übergibt die Übertragung erst anschließend an ein Kommunikationsprogramm. Durch dieses Verfahren kann man Telefongebühren sparen.

Electrosensitive Printer = Elektrosensitiver Drucker
Die elektrosensitiven oder Thermo-Drucker sind billig in der Herstellung, arbeiten leise und ziemlich schnell. Ein Nachteil ist, daß sie nur mit teurem aluminiumbeschichtetem Spezialpapier funktionieren, auf dem die Zeichen oft schwer zu entziffern



Beim Spectrum wird die zu editierende Listingzeile am unteren Bildschirmrand ausgegeben.

sind. Deshalb wird dieses Verfahren kaum für kommerzielle Anwendungen eingesetzt. Für den Ausdruck von Listings sind diese Geräte jedoch ausreichend – ein gutes Beispiel ist der ZX-Drucker von Sinclair, der mit Rollenpapier und nur 32 Zeichen pro Zeile arbeitet.

Die Zeichen werden punktweise aufgebaut. Der Druckkopf ist mit feinen Elektroden bestückt, die unmittelbar vor dem Papier stehen. An diese wird Spannung angelegt. Der resultierende Funke brennt in die Metallisierung auf der Schwarzpapier-Unterlage ein kleines Loch, das als dunkler Punkt erscheint.

Fachwörter von A bis Z

Electrostatic Printer = Elektrostatischer Drucker
Die Bezeichnung „elektrostatisch" wird oft gleichbedeutend mit „elektrosensitiv" verwendet. Tatsächlich arbeiten nur die sogenannten Laserdrucker, die bei Großrechnern verwendet werden, nach dem elektrostatischen Verfahren. Wie bei den Fotokopierern wird bei diesen Geräten zunächst eine Trommel, die mit einem Fotohalbleiter beschichtet ist, elektrostatisch homogen aufgeladen. Dann wird ein Laserstrahl dem Schriftbild entsprechend über die Trommel geführt, wobei sich die Halbleiterschicht an den belichteten Stellen durch Freisetzung von Fotoelektronen entgegengesetzt auflädt. Dort setzen sich anschließend beim Vorbeilaufen der Trommel an der „Tonerstation" feine Farbpartikel fest, die auf Papier übertragen und eingebrannt werden.

Emulator = Emulator
Mit einem Emulator wird auf einem Computer das Verhalten eines Fremdrechners simuliert. Programme für Microcomputer werden oft auf Großrechnern entwickelt, weil dort Speicherkapazität und Testkomfort wesentlich umfangreicher sind. Um beispielsweise Software für den Sinclair Spectrum zu schreiben, braucht man einen Spectrum-Emulator für den Großrechner, der die Eigenschaften des Spectrum simuliert.

ENIAC = ENIAC
ENIAC steht für Electronic Numerical Integrator And Computer – eine universelle Rechenanlage, die in den Jahren 1943–46 von John J. Mauchly und J. Presper Eckert jr. an der University of Pennsylvania/USA entwikkelt wurde und mit Elektronenröhren als Schaltelementen arbeitete. Der ENIAC war ursprünglich für die Berechnung ballistischer Tabellen im Zweiten Weltkrieg gedacht, wurde aber nicht mehr vor Kriegsende fertig. Dieser Computer füllte einen Riesenraum, der wegen der enormen Wärmeabgabe des Rechners mit einer speziellen Klimaanlage ausgestattet war.

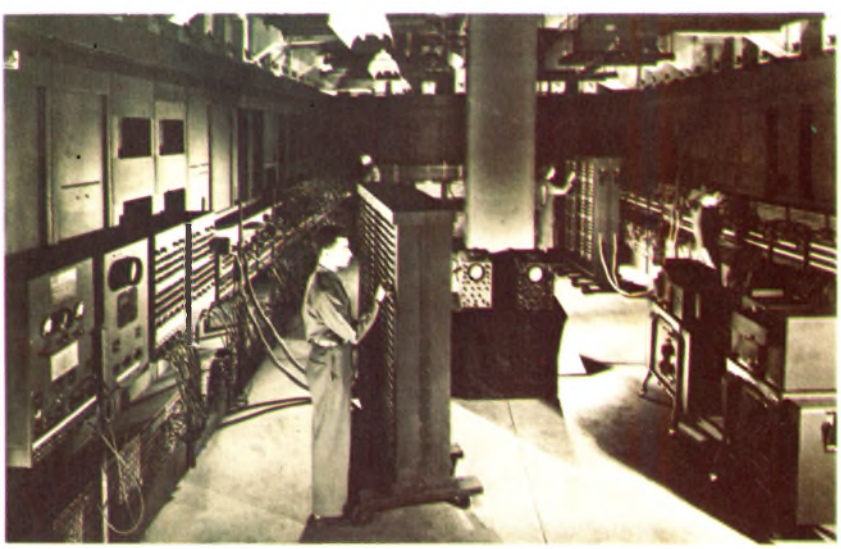
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

EOF = Ende der Datei
An dem Steuerzeichen „End Of File" am Ende einer Datei erkennt das Betriebssystem, daß in diesem File keine weiteren Daten zu verarbeiten sind.

Ergonomics = Ergonomie
Die Ergonomie ist jener Teil der Arbeitswissenschaft, der sich mit der Anpassung von Geräten und Arbeitsplätzen an die physischen und psychischen Bedürfnisse des Menschen befaßt. Die Physiologie spielt bei der äußeren Gestaltung eine wichtige Rolle – ergonomische Gesichtspunkte haben zur Entwicklung frei beweglicher Tastaturen, schwenkbarer Bildschirme, hochauflösender Monitore usw. geführt. Die Psychologie verhilft etwa zu einem Bewußtsein für „Benutzerfreundlichkeit", das heißt zu einer Gestaltung von Hardware und Software, mit der der Benutzer in allen Situationen problemlos zurechtkommt.

Exclusive-OR = Exklusiv-ODER
Die Exklusiv-OR (XOR)-Operation ist eine elementare Funktion der Booleschen Algebra. Sie verknüpft zwei Ein-Bit-Eingänge A und B zu einer Ausgangsgröße C. Deren Wert ist 1, wenn genau ein Eingang die 1 liefert und 0, wenn beide Eingänge zugleich auf 0 oder 1 liegen. Ein XOR-Gatter läßt sich aus zwei AND-Gattern, zwei Invertern und einem OR-Gatter aufbauen. Das XOR gibt es bei vielen Microprozessoren auch als Einzeloperand-Befehl.

Bildnachweise
1065, 1067, 1083, 1085: Chris Stevens
1067: Rolf Seiffe
1069, 1070, 1072, 1075, 1076: Ian McKinnell
1075: Liz Dixon
1077: Tony Sleep
1079: Marcus Wilson-Smith
1082: Dimension Graphics
1086: Liz Heaney
1088: Tony Lodge
1088: „Your 64 and VIC-20"
U3: Science Museum



Dieses Ungetüm mit über 17 000 Röhren und einem Gewicht von 30 Tonnen war die Vorstufe der elektronischen Datenverarbeitung.

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Expert Systems = Expertensysteme
Im Bereich der Künstlichen Intelligenz bezeichnet man Programme für spezielle Wissensgebiete als Expertensysteme. Ein solches System kann unter Rückgriff auf das im Rechner gespeicherte Fachwissen und dessen Auswertung nach vorgegebenen Regeln fachmännischen Rat geben und damit Spezialisten entlasten oder sogar eingeschränkt „vertreten". Ansätze dazu gibt es in der medizinischen Diagnostik und bei der Fehlersuche in mechanischen und elektronischen Systemen und beim CAD.

Exponent = Exponent
Mathematische Gleichungen können oft außerordentlich umständlich werden; deshalb verwendet man bei langen Formeln häufig eine Art Kurzform. Eine Vereinbarung zur Darstellung von Produkten wie

$7x7x7x7x7 = 16\,807$

ist die Exponenten-Schreibweise. Dabei wird das fünfmalige Auftreten des Faktors 7 einfach dadurch ausgedrückt, daß ein Exponent als kleine Ziffer oben rechts neben den Faktor gesetzt wird:

$7x7x7x7x7 = 7^5 = 16\,807$

In der Potenz 7^5 ist 5 der Exponent zur Basis oder Mantisse 7. Bei der Multiplikation ergibt sich das Produkt zweier Zahlen, die als Potenzen zur gleichen Basis darstellbar sind, durch Addition der Exponenten:

$49x343 = 7^2x7^3 = 7^5 = 16\,807.$

Facsimile Transmission = Faksimileübertragung
Die Übertragung von Dokumenten und Bildern spielt in der Bürokommunikation eine wichtige Rolle. Für diese Faksimileübertragung oder Fernkopie gibt es seit vielen Jahren spezielle Geräte. Früher waren das Analogsysteme, bei denen das zu übertragende Schriftstück auf einer rotierenden Trommel befestigt und dann mit einer Fotozelle abgetastet wurde; die Übermittlung einer A4-Seite dauerte damit etwa fünf Minuten. Das elektrische Ausgangssignal der Fotozelle wurde dann über das Telefonnetz an die Empfangsstation gesendet, wo mit dem Thermodruckverfahren aus dem Bildsignal die Vorlage rekonstruiert wurde. Bei neueren Systemen wird das Original mit einem Laser abgetastet und digital an die Empfangsstation übertragen, die mit Hilfe eines Laserdruckers eine Kopie von hoher Qualität erzeugt.

Telefax-Gerät

Factorial = Fakultät
Die Fakultät einer positiven ganzen Zahl(n) ist das Produkt aller ganzen Zahlen von 1 bis n. Als Symbol dafür wird ein Ausrufezeichen hinter die Zahl gesetzt, zum Beispiel:

$1! = 1$
$2! = 1x2 = 2$
$3! = 1x2x3 = 6$
$4! = 1x2x3x4 = 24$

Von der Fakultät wird unter anderem Gebrauch gemacht, um die Anzahl von „Kombinationen" C_n und „Variationen" V_n zu berechnen:

$C_n = \frac{n!}{(n-r)!r!}$ bzw. $V_n = \frac{n!}{(n-r)!}$

Wenn Sie etwa aus einem Packen von 52 Karten an jeden Spieler fünf Blätter austeilen, dann haben Sie C_{52}^{5} verschiedene Kombinationsmöglichkeiten, und insgesamt sind (unter Berücksichtigung der Reihenfolge) genau V_{52}^{5} Variationen (= 311.875.200) beim Auslegen von Fünfergruppen denkbar.

Fail-Safe = Ausfallsicher
Rechner für störungsempfindliche oder risikoreiche Aufgaben werden so ausgelegt, daß sie ausfallsicher sind. Ein vollkommener Schutz gegen alle erdenklichen Fehler ist praktisch unmöglich, aber durch Ersatzsysteme und andere Sicherheitsvorkehrungen kann man gewährleisten, daß etwaige Ausfälle nur eine minimale Auswirkung auf das Gesamtsystem haben.

Ein Beispiel für ein ausfallsicheres Konzept sind die Lokomotiven mit „Totmannknopf": Wenn der Lokführer eine gewisse Zeit nicht auf diesen Hebel gedrückt hat, wird automatisch die Bremse ausgelöst.

Fan-In = Eingangslastfaktor
Der Fan-In gibt an, welche relative Belastung ein angeschlossener Eingang innerhalb eines Schaltkreises darstellt.

Fan-Out = Ausgangslastfaktor
Als Fan-Out wird die Maximalzahl von Gattereingängen bezeichnet, die von einem einzigen Ausgang zuverlässig gespeist werden können. Wird diese Zahl überschritten, so verringert sich die Differenz zwischen den Spannungspegeln für die logische 1 und die logische 0 und damit auch der Störabstand.

Bildnachweise
1093: Mike Brownlow
1094, 1096, 1109, 1118: Ian McKinnell
1095: Roy Ingram
1097, 1098: Kevin Jones
1099, 1110, 1116, 1119: Liz Dixon
1100: Mike Cloews
1101, 1103, 1105, 1106, 1107: Chris Stevens
1112: Adrian Morgan
1114: Liz Heaney

Fachwörter von A bis Z

Feedback = Rückkopplung
Unter Rückkopplung versteht man die Einflußnahme auf ein Steuerungsorgan durch Information über den eingetretenen Zustand – der Wirkungsablauf wird dadurch zum „Regelkreis" geschlossen. Ein einfaches Beispiel ist die thermostatische Regelung eines Warmwasserspeichers. Der Thermostat erfaßt die Wassertemperatur und schaltet die Heizung ein, wenn eine vorgegebene Mindesttemperatur unterschritten wird, und schaltet sie wieder aus, wenn eine bestimmte Höchsttemperatur erreicht ist. Bei digitaler Regelung mit Microprozessoren wird dem Rechner das Rüchkopplungssignal des Meßfühlers über einen Analog/Digital-Wandler zugeführt.

Fibonacci Sequence = Fibonacci-Reihe
Im 13. Jahrhundert entwickelte der Mathematiker Leonardo Fibonacci die Fibonacci-Reihe, eine unendliche Zahlenfolge, in der jede Zahl die Summe der beiden vorangehenden Zahlen darstellt. Die Sequenz beginnt bei 0,1 und setzt sich fort:
1 (die Summe von 0 und 1),
2,3,5,8,13 . . .
Die formale mathematische Definition der Fibonacci-Reihe lautet:
$F_0=0$, $F_1=1$, $F_{n+2}=F_{n+1}+F_n$, $n\geq 0$

Fibre Optics = Glasfaseroptik
Optische Fasern werden aus extrem dünnem Glas oder Plastik hergestellt und lassen sich anstelle von Kupferkabeln zur Übermittlung von Sprache oder Daten über große Distanzen einsetzen. Eine einzige Glasfaser kann mehrere tausend Signale gleichzeitig übertragen. Glasfaserkabel arbeiten nach dem Prinzip der internen Spiegelung: Das Licht bleibt in der Faser, da die äußeren Oberflächen es wieder in das Innere reflektieren. Die Übertragung funktioniert auch bei gedrehten oder stark gekrümmten Fasern. Ein einfacher Versuch mit einem Wasserbehälter macht das Prinzip deutlich. Von einer Lichtquelle im Inneren des Behälters dringt normalerweise kein Licht nach außen. Kann das Wasser

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Demonstration des Glasfaserprinzips

jedoch durch ein Loch abfließen, dann folgt das Licht dem gebeugten Wasserstrahl auf die gleiche Weise wie in der Glasfaser.

Die Informationsmenge, die eine Faser übertragen kann, und die Qualität des Signals hängen von der optischen Dichte des Glases ab. Da optische Kabel keine Elektrizität leiten, können sie gefahrlos in Umgebungen eingesetzt werden, in denen normale Stromkabel ein Sicherheitsrisiko darstellen.

Field = Feld
Felder stellen Informationen dar, die mit einer gemeinsamen Bezeichnung angesprochen werden. In einem Telefonverzeichnis ist normalerweise der Nachname das erste Feld, gefolgt von Vornamen, Adresse und Telefonnummer. Eine Ansammlung von Feldern wird als Datensatz bezeichnet. Die einzelnen darin enthaltenen Daten werden Einträge genannt. Oft dürfen Einträge nur eine bestimmte Anzahl Zeichen enthalten, da ihre Länge in der Felddefinition festgelegt ist.

FIFO = FIFO
FIFO ist die Abkürzung für „First In First Out" und beschreibt eine Methode, die Daten eines Stacks zu bearbeiten. Dabei wird das erste auf dem Stack abgelegte Element auch wieder als erstes heruntergezogen. Stacks können auch in umgekehrter Reihenfolge bearbeitet werden. Dieser Fall heißt LIFO „Last In First Out". Eine FIFO-Liste wird auch Queue List (Warteschlange) oder Pushup-Stack genannt.

File = Datei
Eine Datei enthält gleichartige Informationen, die nach ihrer Erstellung gespeichert, geändert und wiederverwendet werden können. Computerdateien sind normalerweise auf Cassette oder Diskette gespeichert. Sie können vollständige Programme, Programmteile (oft eingesetzte Unterroutinen, die als Modul gespeichert und in anderen Programmen wiederverwendet werden), Daten (die von Programmen geladen werden), Texte (alle Arten von Schriftstücken) oder grafische Daten für visuelle Darstellungen enthalten.

Datenbanken fassen gleichartige Informationen normalerweise in einer Datei zusammen, die als Einheit bearbeitet wird. So würde beispielsweise die Datenbank einer Firma alle Personaldaten in einer separaten „Personaldatei" ablegen.

Bildnachweise
1121, 1123: Ian McKinnell
1124, 1125, 1138, 1139, 1144: Liz Dixon
1126, 1129: Kevin Jones
1128: Crispin Thomas
1133, 1134, 1135: Chris Stevens
1137: Lotus Development Corp.
1141: Liz Heaney
1142, 1143: Michael Brownlow
U3: Nick Daly

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

File Maintainance = Dateipflege
In Dateien gespeicherte Informationen müssen bestimmten Vorgängen unterzogen werden, damit ihre Einsatzfähigkeit erhalten bleibt. Dazu gehört das Anlegen von Sicherheitskopien, das Löschen ungültiger oder nicht belegter Datensätze und das Aktualisieren der gespeicherten Informationen. Diese als Dateipflege bezeichneten Abläufe sind für kommerzielle Daten ebenso wichtig wie für die Programmentwicklung oder eine Datenbank mit einem persönlichen Adressenverzeichnis.

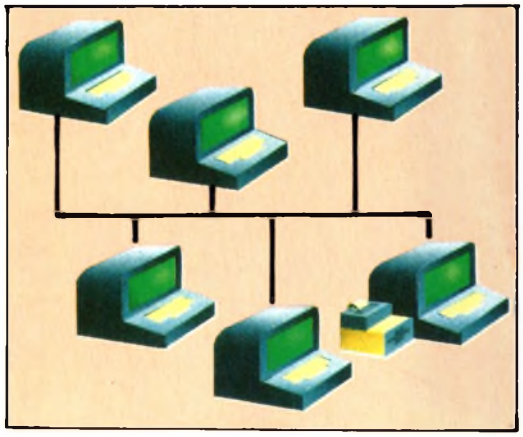
File Protection = Dateischutz
Eine Datei auf Cassette oder Diskette ist im allgemeinen jedem zugänglich, der den Dateinamen kennt. Wenn eine Datei vertrauliche Informationen enthält, etwa die Firmenbuchhaltung oder den Quellcode kommerzieller Software, dann sind Vorkehrungen gegen unberechtigte Zugriffe erforderlich. Am einfachsten ist es, die Datenträger an einem Ort aufzubewahren, der für Unbefugte nicht zugänglich ist. Die Wirksamkeit derartiger Maßnahmen ist aber begrenzt, weil das Material ja für die rechtmäßige Nutzung zugänglich bleiben muß und somit versehentlich in falsche Hände geraten könnte. Bei der Erstellung einer vertraulichen Datei sollte deshalb gleich ein eingebauter Dateischutz vorgesehen werden.

Bei Großanlagen für viele Benutzer fängt die Sicherheit damit an, daß sich jeder mit einer zugeteilten Benutzernummer und einem selbstdefinierten Paßwort „einloggen" muß. Die Benutzernummer bestimmt die „Systempriorität" für den Benutzer, und dazu gehört der Grad der Zugangsberechtigung. Dabei wird beispielsweise jedem Anwender ein bestimmter Directory-Abschnitt zugewiesen, und ein Zugriff auf andere Dateien ist ohne das korrekte Paßwort nicht möglich. Mitarbeiter des Rechenzentrums wie Operateure und Systemprogrammierer haben oft allgemeingültige Benutzernummern – eine Art Generalschlüssel – und damit zu sämtlichen Dateien Zugang.

File Server = File Server
In einem Computer-Netzwerk wird manchmal ein Rechner nur dafür abgestellt, allen anderen Geräten die benötigten Dateien verfügbar zu machen. Zur Erleichterung dieser Aufgabe werden alle Daten von und zu Terminals, Druckern, Diskettenlaufwerken und der sonstigen Peripherie wie unabhängige Dateien behandelt. Der für die Steuerung zuständige Rechner heißt „File Server". Er hat lediglich die Aufgabe, den Datenverkehr zwischen den Netzwerkstationen und den Speichereinheiten zu regeln.

Der File Server wird aktiv, sobald ein Benutzer eine bestimmte Information anfordert – etwa ein Datenfile oder ein Programm auf Diskette oder eine Verbindung mit dem Drucker oder anderweitiger Peripherie. Ist der Kanal für den Benutzer verfügbar, übermittelt der File Server die Daten. Die Datei wird vom Benutzer gegebenenfalls verändert und dann wieder dem File Server überstellt, der sie in der aktualisierten Form zurückspeichert.

File Transfer = Dateiaustausch
Eine Datei wird gewöhnlich im Rechner erstellt und dann in einem peripheren Speicher abgelegt. Die weitere Übertragung auf andere Speicher oder an andere Netzwerkkomponenten, auch an ein entferntes System, stellt einen typischen Fall von Dateiaustausch dar. Schwierigkeiten bei diesem Vorgang bereitet wieder einmal die Kompatibilität. Jedes Gerät hat seine eigenen Spezifikationen hinsichtlich Übertragungsrate, Parität und Steuersignalen. Somit können die Erfordernisse sehr unterschiedlich sein, so daß der Dateiaustausch nicht immer problemlos abläuft.



Beim „Econet" werden bis zu 254 Acorn-Rechner zu einem Netzwerk mit Busstruktur verknüpft. Einem der Rechner wird dann die Rolle des File Server zugewiesen.

Es hat verschiedene Anläufe seitens der Hersteller und diverser Organisationen zu einer Standardisierung gegeben, aber die Probleme sind dadurch eher größer geworden. Das Wachstum der nationalen und internationalen Telefondatennetze erzwingt in letzter Zeit aber doch einheitliche Normen. Innerhalb eines Netzwerks besteht die einfachste Lösung zumeist darin, den ganzen Austausch über denselben Steuerrechner, nämlich den File Server, abzuwickeln, der eine Datei von einer Station übernimmt und dann im entsprechenden Format an eine andere Netzwerkstation weiterleitet.

Bildnachweise
1149: Allan Adler
1150: Ian Dobie
1151: Liz Dixon, Associated Press
1153: Steve Cross
1154, 1155, 1156, 1166: Liz Dixon
1157, 1167, 1169: Ian McKinnell
1158: David Higham
1161, 1162, 1163: Chris Stevens
1170: Soft
1174, 1175: Kevin Jones

Fachwörter von A bis Z

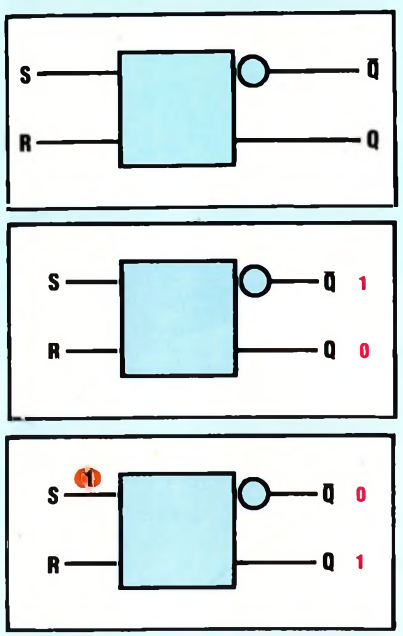
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Filtering = Filterung
Im allgemeinen versteht man unter Filterung die Signalbeeinflussung durch Unterdrückung bestimmter Frequenzanteile. Der bei HiFi-Anlagen gängige Rumpelfilter stellt zum Beispiel einen „Hochpaß" dar, der nur Signalanteile oberhalb einer vorgegebenen Frequenz durchläßt: Beim Aufdrehen des Filters wird diese Grenzfrequenz angehoben, so daß die hochfrequenten Töne bevorzugt werden.

Bei der Übertragung digitaler Signale ist die Rauschfilterung ein wesentliches Hilfsmittel zur Fehlerunterdrückung. Dabei finden Frequenzfilter wie auch Prüf- und Paritätscodes Verwendung. Ferner haben viele Rechner einen Netzfilter zum Fernhalten von Hochspannungsspitzen („Spikes" oder Transienten), die häufig beim Schalten von hohen induktiven Lasten wie Aufzugsmotoren oder großen Kühlaggregaten auftreten. Die Spikes werden vom Spannungsregler des Rechners allein meist nicht aufgefangen und können Resets, unter Umständen sogar Schäden an Bauteilen verursachen.

Eine Filterung (oder „Maskierung") von Information ist durch logische Operationen möglich, indem zum Beispiel nur das jeweils höchstwertige Bit übertragen wird.

Flag = Kennzeichenbit
Ein „Flag" zeigt einen Betriebszustand oder ein Operationsergebnis an. Die CPU verfügt über ein Flag-Register (auch Status- oder Konditionsregister genannt), dessen einzelne Flags je nach Prozessoraktivität und Resultat unterschiedlich gesetzt werden. Wenn ein 8-Bit-Prozessor beispielsweise 236 und 101 addiert hat, wird das Zero-Flag gestrichen, weil das Ergebnis nicht Null lautet. Gleichzeitig wird das Carry-Flag gesetzt, weil die Summe die Kapazität des 8-Bit-Akkumulators übersteigt und ein Übertrag (Carry) erforderlich ist. Einige Assembler-Befehle des Prozessors haben je nach Zustand dieser Flags unterschiedliche Wirkungen, so daß logische Entscheidungen möglich sind.



Die oberste Abbildung zeigt den RS-Flipflop, der ausführlich in unserem Logikkurs beschrieben wurde. Die beiden Ausgänge Q̄ und Q weisen jeweils unterschiedlichen Zustände auf. Das zweite Diagramm stellt den Anfangszustand (RESET) dar, und das dritte verdeutlicht eine Schaltung, wenn der Impuls auf dem SET-Eingang liegt.

Flip-Flop = Flipflop
Da Computer mit Binärzahlen arbeiten, sind Schaltkreise mit zwei stabilen, definiert umschaltbaren Zuständen für den Aufbau von Speicher- und Logikelementen von großer Bedeutung. Solche „bistabilen" Schaltungen werden als Flipflops bezeichnet. Die Grundform ist der RS-Flipflop, der am einfachsten durch kreuzweises Verbinden der Ein- und Ausgänge zweier NAND-Gatter realisiert wird.

Floating Point Notation = Gleitkommadarstellung
Jeder Rechner kennt mindestens zwei Formate für die interne Zahlendarstellung: ganze Zahlen und Gleitkommazahlen. Bei dem ganzzahligen Format werden jeder Variablen zwei Bytes für eine 16-Bit-Zweierkomplementdarstellung zugewiesen. Der Wertebereich ist daher

Dezimal −32 768 bis +32 767
Binär 1000 0000 0000 0000 bis
0111 1111 1111 1111

Im Gleitkommaformat wird eine Zahl dagegen folgendermaßen dargestellt:

$\pm$ Zahl = $\pm$ Mantisse × Basis $^{\pm \text{Exponent}}$

z. B. +317 440 =
+0,60546875 × 2^{+19}

Weil die interne Zahlenbasis bei einem Binärsystem immer 2 ist, braucht sie nicht extra notiert zu werden, so daß nur Mantisse und Exponent mit dem jeweiligen Vorzeichen (Sign = S) zu speichern sind:

	Exponent		Mantisse		
S		S			
0	0010011	0	1001101	10000000	00000000
	Byte 0		Byte 1	Byte 2	Byte 3

Mantisse und Exponent werden als Zweierkomplement-Zahlen dargestellt, so daß das höchstwertige Bit das Vorzeichen enthält (0 für +, 1 für −). Der Exponent wird bei der gebräuchlichen „normalisierten" Darstellung so gewählt, daß die Mantisse nie unter 1/2, aber stets unter 1 liegt.

Der große Vorteil des Gleitkommaformats liegt darin, daß große und kleine Zahlen gleich genau und kompakt darstellbar sind. Mehr Stellen in der Mantisse erhöhen die relative Genauigkeit, mehr Stellen im Exponent erweitern den Bereich.

Bildnachweise
1177: Paul Chave
1178: Liz Dixon, Computer Weekly
1179, 1183, 1187, 1200, 1201: Kevin Jones
1180, 1181, 1204, U3: Ian McKinnell
1184, 1185, 1192, 1193, 1196, 1197: Liz Dixon
1190, 1191, 1194: Chris Stevens
1195: Caroline Clayton
1198: Kevin Jones, Ian McKinnell
1199: Jill Furmanovsky, Time Out

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Floppy Disk = Diskette
Im Laufe der Zeit sind verschiedene Diskettendurchmesser in Gebrauch gekommen: die 8-Zoll-Floppy, die 5 1/4-Zoll-Minifloppy sowie die 3-Zoll- und die 3 1/2-Zoll-Microfloppy. Die Datenträger bestehen aus ferromagnetisch beschichteten, weniger als 1/2 mm starken Kunststoffscheiben. Die Diskette befindet sich in einer Schutzhülle, die mit einem radialen Schlitz („Fenster") versehen ist. Beim Betrieb rotiert die Scheibe mit etwa 300 Upm, während der Schreib/Lese-Kopf über dem Fenster hin- und herfährt. So ist jeder Punkt der Oberfläche schnell erreichbar. Eine Aussparung im Hüllenrand dient als Schreibsperre: Vor dem Beschreiben der Diskette tastet das Laufwerk ab, ob diese Kerbe geschlossen oder frei ist.

Der Schreib/Lese-Kopf arbeitet wie bei einem Magnetbandgerät: Beim Schreiben werden die Magnetisierungsbezirke in der Plattenbeschichtung unterschiedlich ausgerichtet, und beim Lesen wird deren Orientierung abgetastet. Die Information wird auf konzentrischen Spuren (Tracks) aufgezeichnet, die ihrerseits in Sektoren unterteilt sind. Bei Disketten mit „doppelter Dichte" (Double Density = DD) können bis zu 80 Spuren angelegt werden, bei „einfacher Dichte" (Single Density = SD) im allgemeinen 40 Spuren. Die Sektorgrenzen sind bei der sogenannten Hard-Sektorierung durch fotoelektrisch abtastbare Löcher (eins pro Sektor) am Innenrand der Diskette markiert. Bei „Soft-Sektorierung" gibt es dagegen nur ein einziges Indexloch am Anfang des ersten Sektors, und der Beginn jedes folgenden ist nur durch die Magnetisierung gekennzeichnet.

Die Abbildung zeigt eine 5 1/4-Zoll-Diskette, wobei deutlich das Schreib/Lese-Fenster, das Indexloch und die Schreibsperren-Kerbe zu erkennen sind.

Flowchart = Flußdiagramm
Grafische Veranschaulichungen des Ineinandergreifens von Datenoperationen und Steuerungsvorgängen werden häufig als „Flußdiagramm" bezeichnet, obwohl vielfach nur Verknüpfungen statt des Programmflusses dargestellt sind. Ausdrücke wie „Prozeßdiagramm" oder „Datengrafik" wären im Grunde angemessener. Das übliche Flußdiagramm besteht aus Kästchen verschiedener Form, die Programmschritte wie Ein/Ausgabe, Entscheidungen und Wertzuweisungen beschreiben. Diese Kästchen werden durch Flußlinien mit Pfeilen verbunden, die die Abfolge der einzelnen Operationen verdeutlichen. Die meisten Autodidakten benutzen Diagramme dieser Art.

Jedoch verlieren rein grafische Lösungsansätze an Bedeutung gegenüber einer präzisen Festlegung der Anforderungen, der Datenspezifikationen und einer ergänzenden Funktionsbeschreibung.

Flow Control = Datenübertragungssteuerung
Beim Echtzeit-Datenverkehr sind die optimalen Übertragungsraten von Sender und Empfänger meist unterschiedlich. Damit langsame Empfangsgeräte nicht „überfüttert" werden, muß die Übertragung gesteuert werden – entweder vom Sender aus oder von einem Kontrollorgan im Übertragungsnetz. Das einfachste Verfahren ist die „End-to-End"-Steuerung, bei der die Kapazität des Empfängers die Übertragungsrate bestimmt. Eine andere Methode läuft unter der Bezeichnung „Hop-by-Hop" (etappenweise). Dabei wird die Datenrate für jede Teilstrecke des Übertragungsnetzes individuell durch die Kapazität des betroffenen Netzknotens bzw. Übermittlungskanals festgelegt.

Format = Format
In der Datenverarbeitung versteht man darunter im allgemeinen eine Anweisungsstruktur oder die Informationsstruktur auf Datenträgern. Bei Disketten beispielsweise gibt das Format an, wie Spuren und Sektoren auf der Magnetschicht angelegt sind. Aufgrund der unterschiedlichen Formatierung sind auf einem Laufwerk beschriebene Disketten von einem anderen meist nicht lesbar. Mit dem Aufkommen universeller Betriebssysteme wie CP/M und MS-DOS ist die Situation allerdings günstiger geworden, weil Software für diese Systeme oft vom gleichen Diskettenformat Gebrauch macht.

Bildnachweise
1205: Steve Cross
1206, 1210, 1213, 1218, 1219, 1220, 1223: Kevin Jones
1208, 1226: Crispin Thomas
1214, 1215, 1221, 1222: Liz Dixon
1217: Liz Heaney
1227: John Clementson, Dimension Graphics
1231: Ian McKinnell
U3: David Weeks

Fachwörter von A bis Z

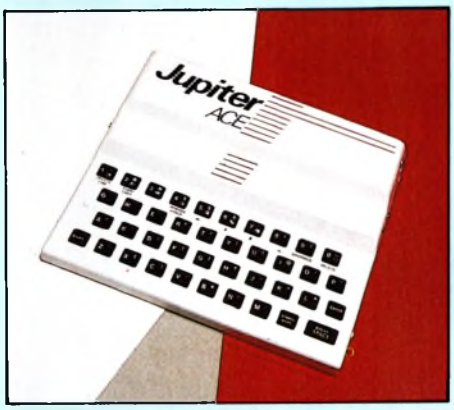
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

FORTH = FORTH
Die Sprache FORTH wurde 1972 von dem Astronomen Charles Moore entwickelt, nachdem er feststellen mußte, daß FORTRAN für seine Teleskop-Steuerprogramme nicht ausreichend war. Die Schwierigkeiten rührten daher, daß Struktur und Verarbeitungsformen bei FORTRAN zu sehr an der mathematischen Lösung wissenschaftlicher Probleme orientiert sind. FORTH bedient sich eines Lexikons von „Primitives" (Stammwörtern) für die elementaren Funktionen und eines Editor/Compiler/Interpreter-Systems.

Der Editor ermöglicht die Definition neuer (unterprogrammähnlicher) Funktionen anhand des vorhandenen Wortschatzes. Die Namen werden dem Lexikon als neue „Wörter" zugefügt. Die Funktionen werden compiliert und gespeichert und sind über den Interpreter jederzeit durch Eingabe des betreffenden Wortes ausführbar. Den RAM-Bereich behandelt FORTH als einen einzigen großen LIFO(Last In First Out)-Stack – den Programmspeicher dagegen als Gebilde aus unabhängigen Abschnitten. Jede Funktion verfügt über eine eigene Unterprogrammadresse. Das Programm springt zur Ausführung auf diese Adresse, greift alle nötigen Parameter vom LIFO-Stack, stapelt die Resultate dorthin zurück und geht zu der nächsten Adresse über.

Beim Programmieren in FORTH kann man für jede Anwendung einen maßgeschneiderten Befehlssatz entwickeln. Die Hauptvorteile sind die Anpassungsfähigkeit und die Geschwindigkeit der Sprache. Der in der Abbildung gezeigte Jupiter Ace wurde mit FORTH als systemeigener Sprache ausgeliefert.

FORTRAN = FORTRAN
FORTRAN (FORmula TRANslator = Formelübersetzer) wurde 1956 als erste kommerzielle Hochsprache bei IBM entwickelt. IBM bewies damit, daß auch ein dem gewohnten Englisch nahestehender Programmiercode schnell und effizient compilierbar sein kann. Die Sprache machte



den Rechnereinsatz vor allem für Ingenieure und andere Wissenschaftler interessant. Sie waren eher bereit, etwas dem vertrauten mathematischen Formalismus Entsprechendes zu erlernen, als die Zeit und Geduld für das Programmieren im Maschinencode aufzubringen. FORTRAN wurde ein durchschlagender Erfolg und ist auch heute noch die verbreitetste Hochsprache.

Als große Errungenschaft galt anfangs die Möglichkeit, Bibliotheken mit unabhängig compilierten FORTRAN-Unterprogrammen aufzubauen. Alle Großrechner verfügen über derartige Programmbibliotheken. Wegen ihrer Bedeutung als „Quellen" wurden andere Sprachen – zum Beispiel PASCAL – so ausgelegt, daß sie auf FORTRAN-Bibliotheksprogramme zugreifen können.

FORTRAN hat eine stolze Zahl von Abkömmlingen, wie ALGOL, PASCAL und BASIC. Das historische Gewicht ist jedoch darin zu sehen, daß FORTRAN den Computer aus den Rechnerlabors der Universitäten herausholte und ihn in Ausbildungsstätten und Fabrikhallen zu einem selbstverständlichen Werkzeug von Wissenschaft und Technik werden ließ. Der Büro- und der Heimcomputer lagen dann nicht mehr fern: FORTRAN machte den Rechner auch dem Nicht-Fachmann zugänglich und leitete einen wichtigen Schritt auf dem Weg zu einer benutzerfreundlichen Datenverarbeitung ein.

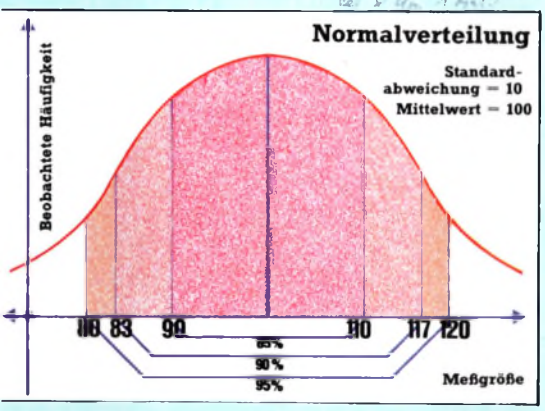
Fourth Generation = Vierte Generation
In der Rechnerentwicklung umfaßt eine Generation immer etwa ein Jahrzehnt. Es beginnt jeweils mit dem Aufkommen einer aufwendigen neuen Technologie, die mit Ende dieser Zeitspanne alltäglich wird. Die Anfänge der „ersten Generation" lagen vor 1950 bei den ersten speicherprogrammierten Rechnern mit Elektronenröhren. Die „zweite Generation" begann Ende der 50er Jahre mit der diskreten Transistorlogik. Die „dritte Generation" vom Anfang der 60er Jahre (typischer Vertreter ist der IBM 360) ist durch die Verwendung integrierter Schaltungen und die Entwicklung umfänglicher Betriebssysteme gekennzeichnet. Und die „vierte Generation" wurde zu Beginn der 70er Jahre mit der Einführung der LSI- und der VLSI-Technik eingeleitet. Diese Schaltungstechnik wird heute bei Großrechnern wie bei Microcomputern angewandt. Auch die Micros haben einige Entwicklungsstadien durchgemacht, bis sie den heutigen Stand als vollwertige Rechner mit beachtlicher Speicherkapazität, integrierter Software und der Fähigkeit zu Mehrprogramm- sowie Verbundnetz-Betrieb erreichten.

Bildnachweise
1233, 1258: Ian McKinnell
1234, 1253, 1256, 1257: Kevin Jones
1236, 1237, 1239, 1240, 1250: Liz Dixon
1238: Wattleworth Silk McConachie, Chris Tucker
1245, 1246, 1247: Chris Stevens
1252: Kevin Jones, Liz Dixon
1254: Liz Heaney

Fachwörter von A bis Z

Frequency = Frequenz, Häufigkeit
Das englische Wort „Frequency" steht sowohl für die physikalisch/technische Größe „Frequenz" wie für den statistischen Begriff der „Häufigkeit" bestimmter Eigenschaften. Die Frequenz eines periodischen Vorgangs gibt an, wie oft ein bestimmter Ablauf – etwa eine elektrische Schwingung – innerhalb der Zeiteinheit stattfindet. Die Frequenz wird in Hertz, abgekürzt Hz, als Anzahl der Zyklen pro Sekunde angegeben. Die Wechselspannungsfrequenz in unserem Versorgungsnetz beträgt 50 Hz; die Musiknote A über dem mittleren C hat 440 Hz; der Prozessor Z80A arbeitet im allgemeinen mit einer Taktfrequenz von 4,25 MHz (entsprechend 4 250 000 elementaren Operationen pro Sekunde), und den NDR II empfangen Sie in Hamburg auf UKW bei 87,6 MHz. Bei der Ausbreitung von Schwingungen in Form von Licht- oder Schallwellen hängt die Frequenz f mit der Wellenlänge λ (Abstand benachbarter identischer Schwingungszustände, etwa zweier Wellenberge) und der Ausbreitungsgeschwindigkeit c zusammen: $c=\lambda * f$

Da die Ausbreitungsgeschwindigkeit elektromagnetischer Wellen (z. B. sichtbares Licht, Infrarot und Rundfunkwellen) circa 300 000 km/s beträgt, errechnet sich für die NDR-Frequenz beispielsweise eine Wellenlänge von 3,42 m.



Von allen Häufigkeitsverteilungen ist die Gauß- oder Normalverteilung in Theorie und Praxis am meisten anzutreffen. Wegen ihrer charakteristischen Form wird die Normalverteilung auch als „Glockenkurve" bezeichnet.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Frequency Distribution = Häufigkeitsverteilung
Wenn Versuchsresultate grafisch so dargestellt werden, daß auf der x-Achse eine veränderliche Meßgröße und auf der y-Achse die zugehörige Häufigkeit aufgetragen wird, ergibt sich ein Bild der „Häufigkeitsverteilung". Dabei muß man die Daten im allgemeinen in Untergruppen zusammenfassen, um eine brauchbare Verteilung zu erhalten. Wenn es sich zum Beispiel um die Häufigkeit gemessener Körpergrößen handelt, lassen sich Gruppen von 161–165 cm, 166–170 cm, 171–175 cm usw. bilden. Die gewonnenen Verteilungen lassen sich mathematisch analysieren und ermöglichen beschreibende Aussagen und die Erstellung von Prognosen.

Häufig werden auch die Begriffe Gauß- oder Normalverteilung erwähnt. Viele menschliche Eigenschaften sind über die Bevölkerung normalverteilt, z. B. Körpergröße, Augenfarbe und (angeblich) auch die Intelligenz. Stichprobenverteilungen nähern sich vielfach der Normalverteilung, wenn der Probenumfang groß genug wird.

Die einzelne Stichprobe ist durch den Mittelwert („durchschnittlicher" Wert der Meßgröße) und die „Standardabweichung" (ein Maß für die Streuung der Daten um den Mittelwert) gekennzeichnet. Bei einer Normalverteilung liegen etwa 65% der Daten weniger als eine Standardabweichung vom Mittelwert entfernt und mehr als 90% weniger als zweieinhalb Standardabweichungen. Die ganze mathematische Statistik fußt größtenteils auf der Analyse von solchen Häufigkeitsverteilungen.

Full Duplex = Vollduplex
Eine Telefonverbindung arbeitet „vollduplex", wogegen der Funksprechverkehr i. a. „halbduplex" abläuft. Der Vollduplexbetrieb läßt eine Datenübertragung in beiden Richtungen gleichzeitig zu, das Halbduplexverfahren erlaubt das nur im Wechsel.

Fuzzy Theory = Fuzzy-Theorie
In digitalen Systemen gibt es nur eindeutige Zustände wie Eins oder Null, Ja oder Nein. Diese elektronikbedingte zweiwertige Logik hat auch die modellmäßige Behandlung der realen Welt in den Rechnerprogrammen geprägt. Damit kommt beim computerorientierten Denken zu kurz, was sonst dem Menschen sehr von Nutzen ist – nämlich die Fähigkeit, mit Wahrscheinlichkeiten und Unsicherheitsfaktoren umzugehen und aufgrund unvollständiger Informationen Entscheidungen zu fällen. Die Fuzzy-Theorie (engl. fuzzy = verschwommen) versucht, dieses Prinzip auf den Rechner zu übertragen. Sie operiert mit einer vielwertigen Logik, in der eine Aussage richtig, wahrscheinlich richtig, vielleicht richtig, wahrscheinlich falsch und falsch sein kann. Dies führt zu unerwarteten Einsichten und bizarren Konstruktionen wie den Fuzzy-Mengen und Fuzzy-Relationen. Die weiteren Forschungsarbeiten über Künstliche Intelligenz dürften auf diesem Gebiet noch interessante Entwicklungen bringen.

Bildnachweise
1261, 1262, 1263, 1269: Chris Stevens
1264: Liz Heaney
1265, 1278, 1285: Ian McKinnell
1266, 1267: Kevin Jones
1273: CAL-Video für Mike Mansfield Enterprises und für BBDO
1274, 1275: D. Lister
1279: David Lawrence
1283: Liz Dixon
1286: Gary Capps-Jenner

Fachwörter von A bis Z

Gate = Gatter
Das Innenleben eines Computers besteht im wesentlichen aus einer Vielfalt elektronischer Schalter. Zur Darstellung der Bitmuster dienen Spannungspegel bzw. sehr kleine elektrische Ströme, und die Verarbeitung der Daten erfolgt, indem für diese Signale durch Öffnen und Schließen der Schalter unterschiedliche Wege freigegeben werden. Wegen dieser Funktion der digitalen Schaltkreise bezeichnet man sie als Gatter.

Vom Typ des jeweiligen Gatters hängt es ab, wie die Informationen verknüpft werden. Die Grundformen sind das AND- und das OR-Gatter, mit denen die entsprechenden logischen Operationen der Booleschen Algebra realisiert werden. Bei der Integration vieler Stufen ist es allerdings einfacher, mit NAND- und NOR-Gattern zu arbeiten.

global = global
Bei einer vergleichsweise einfachen Sprache wie BASIC sind in einem Programmteil verwendete Variablen im allgemeinen auch überall sonst im Programm zugänglich. Wenn etwa zu Beginn X der Wert 134 zugewiesen wird, dann wird bei jedem späteren Zugriff auf X derselbe eingangs mit 134 belegte Speicherplatz angesprochen. Wegen der universellen Gültigkeit spricht man dabei von „globalen" Variablen. Bei umfangreichen Programmen kann es damit aber Probleme geben, besonders bei Teamarbeit oder beim Einbeziehen von Bibliotheksprogrammen. Häufig werden dann einmal eingeführte Variablen irrtümlich für andere Zwecke nochmals verwendet – vor allem, wenn der Compiler nur

Bildnachweise
1289: Marcus Wilson-Smith
1290, 1296, 1302, 1311, 1312, 1313: Ian McKinnell
1291: Ici Plant Protection UK Division
1293: Kawalski, Computer Weekly
1295, 1301: Kevin Jones
1299: Liz Heaney
1300, 1303: Chris Stevens
1310, 1315, U3: Liz Dixon
1316: Roger Sanders

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Schleifenzähler mit einem Buchstaben zuläßt. Einen entscheidenden Nachteil erzeugt die globale Vereinbarung jedoch für die Programmstruktur. Beim Aufruf von unabhängigen Programmteilen wie Subroutinen und Prozeduren sollten nur die als Ein- und Ausgangsparameter angeführten Variablen aktiviert werden. Wenn aber alle Größen global sind, kann natürlich auch jede andere Variable in Mitleidenschaft gezogen werden.

Einige BASIC-Dialekte und viele andere Sprachen wie PASCAL kennen „lokale" Variablen, die nur in dem Programmsegment gelten, für das sie vereinbart wurden. Auch das hat seine Vorteile.

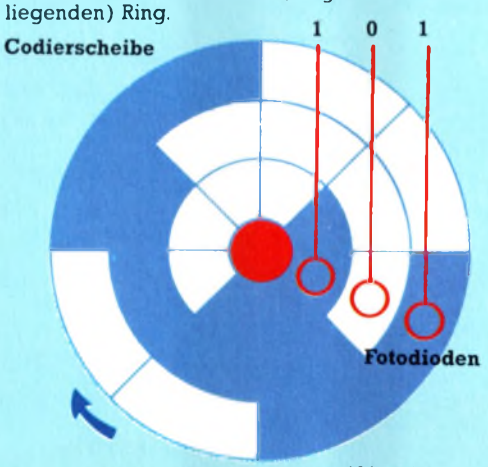
Grandfathering = Generationsprinzip
Das „Grandfathering" ist ein Sicherungsverfahren bei der Aktualisierung von Dateien. Bis zur Fertigstellung der jeweils neuesten Version (Sohn-Datei) wird nicht nur der unmittelbare Vorgänger (Vater), sondern auch der Vor-Vorgänger (Großvater = Grandfather) aufbewahrt. Sollte vor oder bei der Erzeugung der Sohn-Datei durch einen Fehler die Vater-Datei zerstört werden, läßt sich mit Hilfe des noch „lebenden" Großvaters der Vater rekonstruieren und daraus in einem neuen Anlauf der Sohn. Erst wenn der Sohn „fertiggestellt" ist, dürfen Sie die Großvaterdatei überschreiben – ihre Rolle übernimmt nun die Vaterdatei, während die Sohndatei zum Vater aufrückt.

Gray Code = Gray Code
Bei Steuerungsaufgaben muß der Rechner häufig anhand einer mitgeführten Skala die Position bewegter Objekte feststellen, beispielsweise mittels Fotodioden. Erfolgt deren Abfrage gerade dann, wenn der Markierungscode von einem Wert auf den nächsten wechselt, entsteht ein Zufallsergebnis, das unter Umständen Fehler enthält. Beim „Gray Code" sind die Skalenwerte so verschlüsselt, daß die Übergangsfehler minimal bleiben. Das Prinzip besteht darin, bei jedem Einzelschritt nur ein einziges Bit zu ändern, und dessen Stellenwert soll möglichst niedrig sein. Daraus ergibt sich folgende Verschlüsselung:

Dezimal	Binär	Gray Code
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111

Wenn der Sensor zwischen zwei benachbarte Gray-Code-Werte gerät, bleibt der Fehler – anders als beim Binärcode – stets auf eine Einheit der letzten Stelle begrenzt.

Gray-Code-Scheibe
Auf einer rotierenden Welle angebracht, ermöglicht die Scheibe, den Drehwinkel abzulesen. Die drei Ringzonen sind in jedem der acht Sektoren zur Darstellung der Gray-Code-Werte 0–7 unterschiedlich gefärbt; sie werden durch Fotodioden abgetastet. Jeder Sektor unterscheidet sich vom benachbarten nur in einem (möglichst weit außen liegenden) Ring.



101 = 5 (Binär)
101 = 6 (Gray Code)

Fachwörter von A bis Z

Greedy Method = Rabiate Methode
Programme lassen sich unter strategischen oder taktischen Gesichtspunkten entwickeln. Entweder werden vorübergehende Abweichungen von der geraden Linie aus taktischen Gründen toleriert, oder es wird stets der direkte Weg zum Ziel eingehalten. Das letztgenannte Verfahren wird im englischen Sprachgebrauch als „greedy" bezeichnet, womit auch schon die Nachteile dargelegt werden: Unwirtschaftlichkeit und die Unfähigkeit zu differenzieren, nämlich den möglichen Gewinn hinter einem scheinbaren Verlust zu sehen. Ein Schachprogramm beispielsweise, das nur mit direkten Algorithmen arbeitet, verzichtet auf wesentliche Details wie Stellungsspiel und Opferzüge. Aber Schnelligkeit und Agressivität allein haben keine Chance gegen einen Partner, der die taktischen Winkelzüge beherrscht.

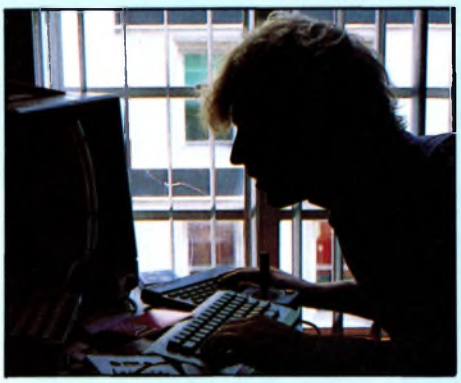
Grosch's Law = Groschs Gesetz
H. R. J. Grosch hat 1953 folgendes Gesetz für die Abhängigkeit der Leistungsfähigkeit eines Computers vom Preis verfaßt:

Leistung = const. * $(\text{Preis})^2$

In den fünfziger Jahren wurde diese Formel viel zitiert. Ein Computersystem, das dreimal soviel kostet wie ein anderes, müßte danach neunmal soviel leisten. Inzwischen hat die hochintegrierte Schaltungstechnik Groschs Gesetz jedoch vollständig unterminiert.

Hacker = Hacker
Ein Hacker ist im Fachjargon ein computerbegeisterter Mensch, der viel Zeit und Mühe investiert, um Schutzroutinen, Betriebssysteme und (via Telefon) fremde Computersysteme zu knacken. Die Hacker neigen zu einem großzügigen Umgang mit besitzrechtlichen Fragen, in einigen Fällen führt das bis zur Computerkriminalität. Die große Mehrheit beschränkt sich aber darauf, die Grenzen des eigenen Könnens bzw. des eigenen Rechnersystems zu erkunden. Ins Licht der Öffentlichkeit rückten die Hacker unter anderem durch den Film „War Games", in

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



Die Ingenieurstudenten vom Massachusetts Institute of Technology machen sich angeblich alljährlich den Spaß, im Kontrollsystem eines Bostoner Bürohochhauses herumzuhackern, so daß dort das Licht stundenlang an- und ausgeht. Aber es bleibt nicht immer bei harmlosen Spielchen, und die Computerkriminalität steigt ständig.

dem ein einfallsreicher Amateur durch Einbruch in das Rechnernetz der US-Abwehr um ein Haar den dritten Weltkrieg ausgelöst hätte.

Half Duplex = Halbduplex
Ein Beispiel für den Halbduplex-Betrieb sind die Funkanlagen in Taxis und Einsatzfahrzeugen: Die Übertragung erfolgt in beiden Richtungen auf dem gleichen Kanal, der abwechselnd freigegeben wird.

Hamming Codes = Hamming Codes
Bei der Fernübertragung von Daten besteht immer die Gefahr, daß sich Fehler einschleichen. Deshalb haben die Informatiker eine ganze Reihe von Fehlersuch- und Fehlerkorrekturverfahren entwickelt. Für die Korrektur einzelner Bitfehler in binären Informationsblöcken stellen die „Hamming Codes" (um 1950 von R. W. Hamming bei der Firma Bell Telephone entwickelt) eine perfekte Lösung dar.

Angenommen, Sie wollen den Datenblock 0111 übertragen. Im Hamming-Code werden drei weitere Bits hinzugefügt, die so definiert sind, daß bestimmte Viererkombinationen aus den insgesamt sieben Bits immer eine gerade Anzahl von Einsen enthalten. Im obigen Beispiel müßte der Hamming-Code 100 lauten, der ganze Sieben-Bit-Block somit 0111100. Die Hamming-Kombinationen sind:

Datenwort 0111100		
Kombinationen	**Anzahl v. Einsen**	**Log. Resultat**
0.1.1.0	gerade	0
.11..00	gerade	0
...1100	gerade	0

Das logische Gesamtergebnis ist „000", gleichbedeutend mit „fehlerfrei". Stellen Sie sich nun vor, bei der Übertragung sei das dritte Bit von links „gekippt":

Datenwort 0101100		
Kombinationen	**Anzahl v. Einsen**	**Log. Resultat**
0.0.1.0	ungerade	1
.10..00	ungerade	1
...1100	gerade	0

Das logische Ergebnis 011 (binär) = 3 (dezimal) läßt erkennen, daß das dritte Bit von links umgefallen ist. Diesen Fehler kann die Empfangsstation automatisch korrigieren. Das Verfahren versagt nur, wenn mehr als eins von den sieben Bits fehlerhaft ist.

Bildnachweise
1317, 1340, 1341: Chris Stevens
1318, 1319: Kevin Jones, Caroline Clayton
1320, 1321, 1335, U3: Ian McKinnell
1323, 1324, 1328, 1332, 1333, 1339: Kevin Jones
1325, 1326, 1327: Liz Dixon
1330: Graphic Products, London
1331: Magnenat-Thalmann, Thalmann – University of Montreal, Prince – Genigraphics
1336: BBC Hulton Picture Library
1337: Uta Brandl
1342, 1344: Janet Barrance

Fachwörter von A bis Z

Handshaking = Quittungsbetrieb
Ohne exakte zeitliche Abstimmung der Operationen funktioniert kein Rechner. Auch altgediente Microprozessoren wie der Motorola 6800 arbeiten schon mit einer Taktfrequenz von 1 MHz, das heißt, sie benötigen für einen Einzelschritt wie die Befehlsübernahme aus dem RAM etwa eine Mikrosekunde. Viele Peripheriegeräte sind dagegen wesentlich langsamer – auch ein schneller Matrixdrucker beispielsweise braucht mindestens 3000 Mikrosekunden pro Zeichen. Für den Datenverkehr zwischen Rechner und Peripherie sind daher strikte Vereinbarungen im sogenannten „Protokoll" erforderlich. Beim Handshake-Betrieb ist für die Verzahnung der Abläufe in der Form gesorgt, daß ein Gerät dem anderen zunächst nur seine Sende- oder Empfangsabsicht mitteilt und erst aktiv wird, nachdem der Partner seinerseits mit einem Ready-Signal geantwortet hat. Wenn die Übertragung nicht so geregelt wird, kommt es zu Fehlern, weil schnellere Empfänger Daten doppelt lesen oder zu langsame Empfänger nur die ersten Bits der eintreffenden Information mitbekommen.

Der Handshake-Betrieb läßt sich rein softwaremäßig, aber auch hardwaremäßig innerhalb einer Schnittstelle realisieren. Beim „Peripheral Interface Adaptor" (PIA) 6820 von Motorola zum Beispiel erfolgt die Kommunikation über ein Buffer-Register. Sobald dort Daten anstehen, wird im Statusregister des PIA ein Flag gesetzt, das erst nach dem Auslesen des Datenregisters wieder gelöscht wird. Damit ist ohne weiteres ein Handshake-Betrieb möglich: Wenn etwa die CPU ein Zeichen an den PIA gesendet hat, fährt sie zunächst mit der Programmbearbeitung fort, bis das Flag im PIA-Statusregister zurückgesetzt ist, das heißt, bis das angesprochene externe Gerät die Daten übernommen hat. Dann erst kann die CPU das nächste Zeichen abschicken.

Bildnachweise
1345: Tony Duncan-Smith
1347: Steve Cross
1348, 1349, 1366: Ian McKinnell
1350, 1351, 1353, 1356, 1361, 1367, 1370: Kevin Jones
1357, 1358, 1359: Chris Stevens
1360: Bernard Jennings
1364, 1365: David Higham
1369: Liz Heaney
U3: David Weeks

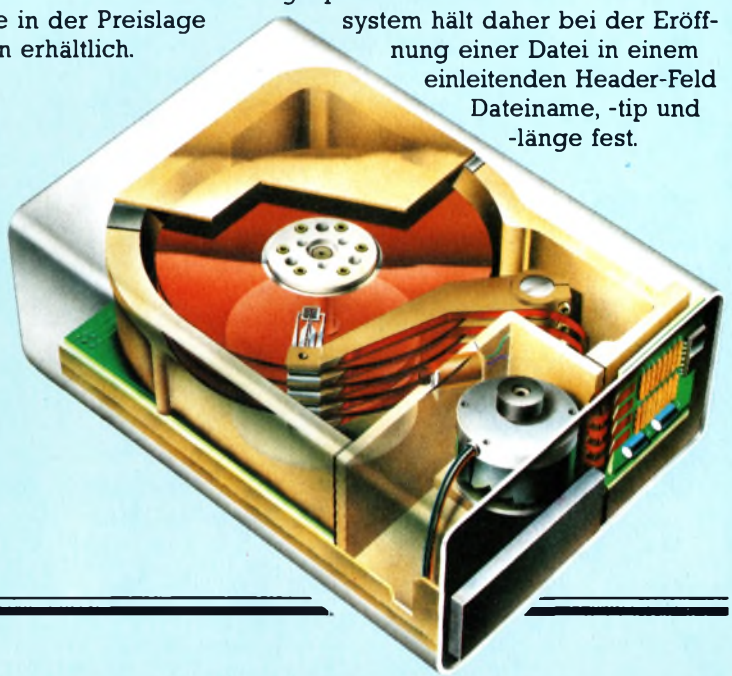
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Hard Disk = Festplatte
Als Alternative zu den Diskettenlaufwerken, bei denen das Speichermedium zwar austauschbar ist, aber nur geringe Kapazität und Geschwindigkeit zuläßt, erfreuen sich die schnellen Festplatten mit ihrer großen Aufnahmefähigkeit wachsender Beliebtheit. Die starren Platten rotieren mit hoher Drehzahl in einem versiegelten Gehäuse. Das ermöglicht rund die zehnfache Aufzeichnungsdichte wie bei Disketten. Heute sind Festplatten mit Kapazitäten zwischen 10 und 30 Megabyte in der Preislage von Bürorechnern erhältlich.

Der Umgang mit den Hard Disks erfordert einige Sorgfalt, speziell hinsichtlich der Sicherung von Betriebssoftware und Datenfiles.

Hashing = Hash-Codierung
Wenn große Datenmengen mit System in begrenztem Speicherraum untergebracht werden sollen, braucht man ein Zuordnungsverfahren, das an der Datenträgerstruktur orientiert ist. Eine Möglichkeit ist die Hash-Codierung, die eine rationelle Raumnutzung bei erträglichen Zugriffszeiten erlaubt. Angenommen, Sie möchten die Datensätze in einer Datei alphabetisch anordnen, wobei der Text im jeweils ersten Datensatzfeld wie bei einem Lexikon die Reihenfolge bestimmen soll. Vorteilhaft wäre es, unmittelbar die ASCII-Codes zur Adressierung der Datensätze heranzuziehen. Die Anzahl der möglichen Kombinationen aus nur zehn ASCII-Zeichen, von denen jedes 26 verschiedene Werte (von A–Z) annehmen kann, ist aber so groß, daß sie in keine Datei hineinpassen würde. Die Lösung besteht darin, die ASCII-Zeichenfolgen durch „Hashing" so zu verschlüsseln, daß sich die Anzahl von Kombinationsmöglichkeiten reduziert.

Header = Dateikopfsatz
Daten und Programme können als unterschiedlich formatierte Daten gespeichert werden. Das Betriebssystem hält daher bei der Eröffnung einer Datei in einem einleitenden Header-Feld Dateiname, -tip und -länge fest.



Heute ist es üblich, jede Festplatteneinheit „Winchester"-Laufwerk zu nennen – ursprünglich war das eine von IBM geprägte Bezeichnung. Erzählungen zufolge hießen dort die Festplatten in der Entwicklung wegen ihrer Speicherkapazität von 2 × 30 Megabyte zunächst 30/30er, was zufällig das Kaliber der historischen Winchesterbüchse war.

Fachwörter von A bis Z

Hertz = Hertz
Der Begriff Hertz, benannt nach dem Physiker Heinrich Hertz, ist eine Maßeinheit für die Frequenz. Wenn sich irgendein Vorgang einmal pro Sekunde wiederholt, hat er eine Frequenz von einem Hertz. Die Einheit wird in der Akustik für die Tonhöhe ebenso verwendet wie in der Nachrichtentechnik für periodische elektrische Signale – zum Beispiel gibt die Zeilenfrequenz die Häufigkeit der horizontalen Abtastung des Bildschirms durch den Elektronenstrahl an.

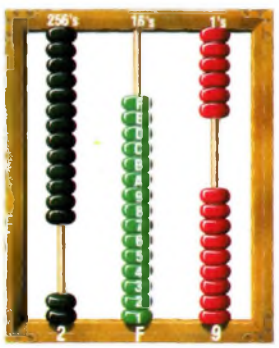
Das Hertz wird mit Hz abgekürzt; höhere Frequenzen gibt man in Kilohertz (kHz) oder in Megahertz (MHz) an.

Heuristic = Heuristisch
Bei heuristischen Systemen wird die Lösung von Problemen rein empirisch angegangen, also in Form des Lernens aus Versuchsansätzen. Nicht-heuristische Systeme gehen statt dessen nach einem festen Bearbeitungsschema vor, das auf der Basis des existierenden Wissens erstellt worden ist. Bei einfachen Gleichungen beispielsweise findet man die Unbekannte durch eine Reihe mathematischer Umformungen, die insgesamt den „Lösungsalgorithmus" bilden. In allen Fällen, wo das Ziel durch einen bereits bekannten oder aufgrund von Vorkenntnissen eigens aufgestellten Algorithmus in gerader Linie angesteuert wird, handelt es sich um ein nicht-heuristisches Verfahren.

Ein heuristisches System folgt dagegen nur bis zu einem gewissen Punkt definierten Vorgaben und sucht sich dann selbst den besten Lösungsweg in einem Lernprozeß, im allgemeinen durch Ausprobieren verschiedener Möglichkeiten und Rückmeldungen der Resultate. Auch bei Computern werden heuristische Verfahren angewandt, wenn Entscheidungen zu fällen sind, für die keine ausreichende Basis gegeben ist.

Hilfsalgorithmen können den Lernvorgang in vielen Punkten unterstützen, aber nicht ersetzen.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



Das Rechnen im Hexadezimalsystem würde einen Abakus (Rechenrahmen) mit 15 Perlen je Stange erfordern. Die Perlen entsprechen dabei den Dezimalzahlen 0–15, die hexadezimal durch die Ziffern 0–9 und die Buchstaben A–F dargestellt werden. Der rechte Stab ist für die Einer, der mittlere für die Vielfachen von 16 und der linke für die Vielfachen von $16 \times 16 = 256$ – anders ausgedrückt: rechts hat jede Perle den Wert 1, in der Mitte 16 und links 256. Die Abbildung zeigt die Dezimalzahl 761 (von links her gerechnet: $2 \times 256 + 15 \times 16 + 9 \times 1$), zu lesen als „2F9 Hex".

Hexadecimal = Hexadezimal
Die hexadezimale Zahlendarstellung bedient sich der Basis 16, so wie das Binärsystem mit der Basis 2 oder das Dezimalsystem mit der Basis 10 arbeitet. Die Hexadezimal (kurz Hex)-Schreibweise kennt außer den Ziffern 0–9 noch die Buchstabensymbole A–F für die Zahlenwerte 10–15, so daß der Bereich von 0-15 einstellig notiert werden kann.

Vom Hex-System wird bei der Assemblerprogrammierung ausgiebig Gebrauch gemacht. Die Zahl 255 etwa belegt dezimal drei Stellen und in Binärform als 11111111 sogar acht, während der Hex-Code FF nur zwei Stellen beansprucht.

Einige Microcomputer haben für die hexadezimale Eingabe eigens einen Hex-Tastenblock, bestehend aus 16 Tasten mit den Bezeichnungen 0–9 und A–F.

Hierachical Communications System = Hierarchisches Kommunikationssystem
Als hierarchisch wird ein Kommunikationsnetz dann bezeichnet, wenn es nach Kompetenzebenen gegliedert ist. Die unterste Ebene hat eine sehr begrenzte Zuständigkeit, während jede höhere Ebene mehr und mehr übergreifende Verantwortung besitzt und auf Information aus den Schichten darunter angewiesen ist. Das läßt sich am besten anhand der Organisation eines großen Unternehmens mit weiträumig verstreuten Agenturen verdeutlichen: Jede Bezirksdirektion hat eine Abteilung für die Abwicklung der Aufträge aus dem eigenen Revier, mit einem lokalen Kommunikationssystem, das den Kontakt zwischen den Ortsagenten herstellt (unterste Ebene). Für die Vorgänge innerhalb des Bezirks und für den regionalen Netzknoten ist ein Vertriebsleiter verantwortlich. Dessen Büro kommuniziert über Fernverbindungen mit den Vertriebsleitern aller anderen Bezirke (mittlere Ebene), außerdem mit der Auftragszentrale der Firma. Dort sitzt der Vertriebsdirektor des Unternehmens, der Zugriff zu den Regionalinformationen hat und seinerseits mit den übrigen Abteilungschefs der Firmenleitung verbunden ist (oberste Ebene).

Bildnachweise
1373: Paul Chave
1374, 1383, 1384: Kevin Jones
1375, 1378, 1379, 1388, 1392, 1393: Ian McKinnell
1381: Liz Dixon
1382: Ian McKinnell, Liz Heaney
1387: G. Gardner, J. Pasquale
1395: Aircall, Ian McKinnell
1397: Uta Brandl
1398: Helga Völker
U3: Mark Watkinson

Fachwörter von A bis Z

Hi-Res Graphics = Hochauflösende Grafik
Hi-Res ist die Abkürzung für High Resolution (hohe Auflösung) – damit bezeichnen die meisten Hersteller die höchste Grafik-Auflösung, die bei ihrer Maschine vorgesehen ist. Aus wie vielen Bildpunkten (Pixeln) die Grafik im Hi-Res-Betrieb tatsächlich aufgebaut wird, hängt also vom Rechner ab. Mit der Anzahl der Pixel nimmt die Auflösung zu, und das Bild wird detailreicher und brillanter. Spezielle Computergrafik-Systeme erreichen eine horizontale und vertikale Auflösung von mehreren tausend Punkten.

Hollerith Code = Hollerith-Code
Herman Hollerith (1860–1929) entwickelte im Jahr 1888 die heute noch gebräuchliche Verschlüsselung von Buchstaben, Ziffern und Zeichen auf Lochkarten. Die Karte ist in 12 Zeilen mit je 80 Stanzpositionen aufgeteilt, wobei in jeder senkrechten Spalte durch ein bis drei Lochungen ein einzelnes Zeichen dargestellt wird. Die Information ist durch Kartenleser abtastbar und wird von „Tabelliermaschinen" mechanisch verarbeitet.

Für die Auswertung der amerikanischen Volkszählung entwickelte Herman Hollerith 1888 das Lochkartenverfahren, das sich später auch in Büros durchsetzte. Er gründete eine Gesellschaft zur Herstellung von Lochkartengeräten („Tabelliermaschinen"), aus der später die Firma IBM hervorging.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Holographic Memory = Holografischer Speicher
Als neueste Errungenschaft bei den Kredit- und Scheckkarten gilt die Fälschungssicherung durch ein eingeprägtes „Hologramm" – ein Interferenzmuster, das fotografisch mit Hilfe eines Laserstrahls erzeugt wird. Die Banken und Kreditkartengesellschaften hoffen, damit Betrügern das Handwerk zu legen, weil das Hologramm sehr schwer nachzuahmen ist. Viel interessanter als diese spezielle Anwendung sind aber eigentlich die Perspektiven, die sich für den Einsatz holografischer Systeme als Massenspeicher am Computer abzeichnen.

In den Entwicklungslabors existieren holografische Speicher schon seit Anfang der siebziger Jahre. Die binäre Information wird dabei als Hologramm verschlüsselt auf einer Fotoschicht festgehalten. Durch Projektion des Hologramms mit einem schwachen Laser lassen sich die Ausgangsdaten rekonstruieren. Holografische Speicher sind noch mehr als Laserplatten gegen Umgebungseinflüsse wie Staub und Temperaturschwankungen unempfindlich, ebenso gegen oberflächliche Kratzer. Ende der siebziger Jahre gelang es bereits, 200 Millionen Bits auf einer Kunststoffkarte von 10 × 15 cm unterzubringen.

Host Computer = Host-Computer
Als Host-Computer wird bei Verbundsystemen der Computer bezeichnet, der für die Ablauforganisation zuständig ist. Dazu gehören die verschiedensten Aufgaben. In einem Local Area Network sind es im wesentlichen Server-Funktionen: Der Host-Computer stellt Dateien bereit, steuert die Datenübertragung und erledigt das Ausdrucken für alle anderen Systemkomponenten. Bei leistungsfähigeren Anlagen, vor allem beim Großrechner-Verbund, kann der Host-Computer die Verwaltung des Time-Sharing übernehmen. In einem hierarchischen Kommunikationssystem mit Rechnern unterschiedlicher Rangstufen dient ein Computer unter Umständen in der einen Ebene als Host-Computer und bearbeitet gleichzeitig auf einer anderen Ebene selbst Programme.

Human Factors Engineering = Menschengerechte Gestaltung
Weil der Rechner von Menschen bedient werden soll, erfordert die konstruktive Gestaltung (Engineering) die Berücksichtigung von auf dem Menschen beruhenden Einflußgrößen. Das „Human Factors Engineering" zielt darauf, perfekte Mensch-Maschine-Systeme zu schaffen, indem die traditionelle technikorientierte Vorgehensweise mit arbeitspsychologischen und sozialwissenschaftlichen Ansätzen zum Gesamtkonzept verschmolzen wird.

Computer-Neulinge haben vielfach Hemmungen, die der Gestalter mit einem benutzerfreundlichen Konzept überwinden kann. Derartige Schritte sind auf der Softwareseite zum Beispiel Menütechnik und Dialogsprachen, auf der Hardwareseite „Benutzerschnittstellen" wie die Maus als Eingabegerät oder der berührungsempfindliche Bildschirm. All dies sind aussichtsreiche Versuche, die Furcht vor dem Computer abzubauen. Auch Gesichtspunkte wie die Anordnung von Arbeitsflächen und die Aufstellung der Rechner fließen in eine Gesamtkonzeption ein.

Bildnachweise
1401, 1408, 1409, 1410, 1421, 1422: Kevin Jones
1407, 1411, 1428: Ian McKinnell
1412: Liz Heaney
1413, 1426, 1427: Chris Stevens
1417, 1424, 1425: Liz Dixon

Fachwörter von A bis Z

Identification = Benutzeridentifikation
In Rechnernetzen muß die Identität des Benutzers überprüft werden, um einen unberechtigten Zugriff zu vertraulichen Informationen zu verhindern. Die Identifikation erfolgt meist in der Form, daß der Benutzer sich mit Namen und Benutzernummer beim System anmeldet („einloggt") und zusätzlich ein Paßwort eintippt, das ihm den Zugang freigibt.

Weil sich gezeigt hat, daß dieses Verfahren keinen ausreichenden Schutz gegen Mißbrauch durch Hakker bietet, bemühen sich Benutzer wie Hersteller um zuverlässigere Maßnahmen. Dr. Kuno Zimmerman von der Universität Missouri-Columbia (USA) machte zum Beispiel den Vorschlag, daß der Benutzer auf einem kleinen Grafiktablett seinen Namenszug hinterlassen sollte, dessen charakteristische Merkmale dann mit dem im Rechner gespeicherten Original verglichen werden. Es gibt Unterschriftenfälscher ...

Handschriftliche Mitteilungen, auch Unterschriften zur Benutzeridentifikation, können mit einem Telefaxgerät telefonisch an die jeweilige Empfängerstation übermittelt werden. Die Übertragung erfolgt geräuschlos und verschlüsselt, so daß Abhörversuche sinnlos sind.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

IEEE = IEEE
Das amerikanische „Institute of Electrical and Electronic Engineers" ist 1963 aus dem „Institute of Radio Engineers" und dem „American Institute of Electrical Engineers" hervorgegangen. Dieser Verband mit weltweit über 200 000 Mitgliedern gibt technische Normen heraus.

Als interessierter und gestandener Heimcomputerbesitzer hatten Sie möglicherweise schon mit dem IEEE-488-Bus zu tun – ein verbreiteter Parallelschnittstellen-Standard für die Peripherie. Diese IEEE-Norm (gesprochen: „I triple E") schreibt vor, daß acht Leitungen für die byteweise Parallelübertragung der Daten vorliegen müssen, so viele Adreßleitungen, wie die höchste Systemadresse erfordert, und Steuerleitungen für Ein- und Ausgabe. Auch Quittungsbetrieb (Handshaking) ist vorgesehen, um eine korrekte Übertragung sicherzustellen. Der IEEE-Bus ist unter anderem zum Datenaustausch zwischen unterschiedlichen Rechnern geeignet. Für den Computerfreak also notwendig.

IF-THEN-ELSE = IF-THEN-ELSE
Die bedingte IF-THEN-Anweisung gehört zum Wortschatz fast aller Heimcomputer. Die ELSE-Erweiterung kennen leider nur die anspruchsvolleren Modelle. Das vollständige IF-THEN-ELSE ermöglicht die direkte Programmierung einer echten Verzweigung: Wenn die zum IF gehörige Bedingung erfüllt ist, führt der Rechner die auf THEN folgende Anweisung aus. Andernfalls springt er auf die mit ELSE eingeleitete Alternative. Diese kann beispielsweise im Aufruf eines Unterprogramms bestehen, das seinerseits weitere Bedingungen abfragt.

Häufig ist nur das IF-THEN zugelassen. Dann bearbeitet der Rechner bei nicht erfüllter Bedingung direkt die nächste Programmzeile, die er andernfalls auf dem Umweg über die THEN-Anweisung auch erreicht. Für eine echte Verzweigung müßten hier weitere IF-Befehle verwendet werden. Das ist umständlicher und macht die Fehlersuche schwieriger, als es bei der klaren Trennung mit IF-THEN-ELSE der Fall ist.

Impact Printer = Anschlagdrucker
Bei Anschlagdruckern entsteht das Schriftbild oder die Grafik durch mechanischen Anschlag von Druckwerkzeugen auf ein Farbband. Zu unterscheiden ist im wesentlichen zwischen Ganzzeichen- und Matrixdruckern. Ganzzeichendrucker erzeugen durch einmaligen Anschlag einer Relieftype gleich das vollständige Symbol, während bei Matrixdruckern jedes Zeichen aus einzelnen Punkten aufgebaut wird.

Der Druckkopf der Matrixgeräte enthält dazu eine Anzahl von Anschlagnadeln, die elektromagnetisch gegen Farbband und Papier geschleudert werden, so wie es das Zeichen erfordert. Auch Schreibmaschinen gehören im Prinzip zu den Anschlagdruckern, werden aber normalerweise nicht dazu gerechnet, weil die Ansteuerung über die eigene Tastatur erfolgt.

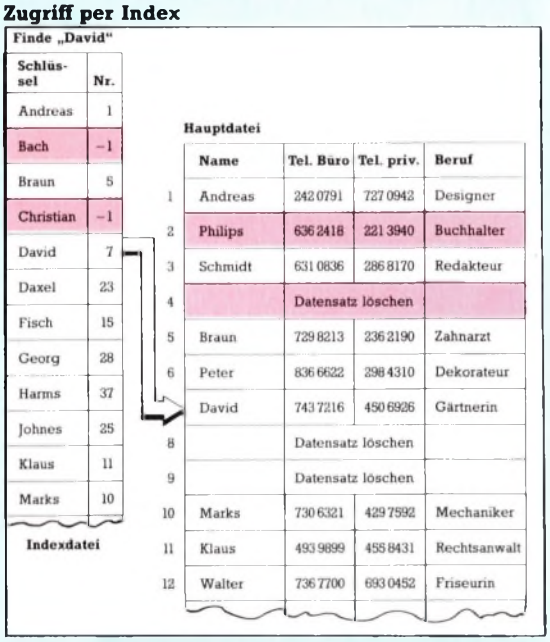
Bildnachweise
1429, 1441, 1442, 1443: Chris Stevens
1431, 1451, 1453: Kevin Jones
1435: Ian McKinnell
1439: Mike Clownes, Uta Brandl
1444: Marcus Wilson-Smith
1445: Tony Sleep, Computer Recognition System
1447: Caroline Clayton
U3: Steve Cross

Fachwörter von A bis Z

Index = Index
Das Wort Index wird in der Datenverarbeitung in mehrfachem Sinn gebraucht. Häufig ist darunter die Positionsnummer einer Informationseinheit innerhalb einer längeren Liste zu verstehen. Die Indizierung ermöglicht den Einzelzugriff, indem lediglich nach dieser Nummer gesucht wird. Die Feldelemente eines BASIC-Arrays werden zum Beispiel über ihren individuellen Feldindex angesprochen. Bei der Maschinencode-Programmierung spielt das Indexregister eine wichtige Rolle. Vielfach wird auch die Schleifenzähler-Variable bei zyklischen Programmabschnitten als Index bezeichnet.

Indexed File = Index-organisierte Datei
Wenn eine Datei mit einem Index organisiert wird, so wird dieser oft wie der separate Index eines Buches völlig von der Datei getrennt. Ein vertrautes Beispiel ist die Verwaltung von Disketten, bei der der Index als Directory auf einer Extraspur

Zugriff per Index



Die Indexdatei enthält die Schlüsselfelder der Datensätze in der gleichen Reihenfolge, wie sie in der Hauptdatei stehen. Zum Aufsuchen eines bestimmten Datensatzes wird in der Indexdatei die Position des Schlüssels festgestellt. Zum kurzfristigen Löschen einzelner Sätze genügt ein Vermerk in der Indexdatei.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

gespeichert wird. Vor jedem Zugriff wird zunächst die Directory befragt, wo die gesuchte Datei auf der Diskette abgelegt ist.

Ein besonders wichtiger Fall ist die indizierte sequentielle Datei. Hier wird zunächst eine sequentielle Datei angelegt. Die zugehörige Indexdatei enthält nur die Schlüsseldaten der Datensätze, und zwar in derselben Reihenfolge. Zum Aufsuchen eines Datensatzes wird erst die Indexdatei nach den gewünschten Schlüsseldaten durchsucht, und anhand der zugehörigen Positionsziffer kann dann in der sequentiellen Datei direkt auf den richtigen Datensatz zugegriffen werden. Dieses Verfahren wurde ursprünglich für die Bandspeicher von Großrechenanlagen entwikkelt, da die sequentielle Datei selbst wegen ihres Umfangs oft nicht mehr in den Arbeitsspeicher paßt. Die Indexdatei läßt sich jedoch auf kleinstem Raum unterbringen.

Index Register = Indexregister
In der Zentraleinheit eines Microprozessors sind ein oder mehrere Ein-Byte-Register für das Speichern der vom Programm benötigten Indizes vorgesehen, insbesondere zur Unterstützung der indirekten Adressierung. Dabei wird der Inhalt eines Indexregisters zur Operandenadresse eines Befehls addiert, und der Zugriff erfolgt dann auf die berechnete neue Adresse. Zum Beispiel bewirkt LDA BASE,X das Laden des Akkumulators mit dem Byte, dessen Adresse sich als Summe der Basisadresse BASE und des Index im Register X ergibt.

Die indirekte Adressierung erleichtert vor allem den Umgang mit Arrays – unter Verwendung spezieller Indexbefehle für die Adreßberechnung. Ein Indexregister läßt sich aber auch zum Ablegen kurzfristig benötigter Variablen benutzen. Damit sparen Sie Speicherplatz und Rechenzeit. Schreiben und Lesen dauert länger als ein Rasterzugriff.

Information Management System = Informationssystem
Ein Informationssystem hat die Aufgabe, Informationen verschiedenster Art anzubieten und die Hilfsmittel zu deren Verknüpfung und Auswertung bereitzuhalten. Sehr häufig handelt es sich dabei um Datenbanksysteme. Es genügt nicht, die Information nur vorrätig zu haben, sondern der Benutzer muß darauf auch leicht und schnell zugreifen können, zum Beispiel über einen Suchindex. Ein vollwertiges Informationssystem ist in Verbindung mit der dezentralen Datentechnik (Distributed Processing) ein außerordentlich wirkungsvolles Werkzeug.

Der Begriff Information Management System wird oft auch allgemein für die Organisationsprogramme eines Systems verwendet, zum Beispiel für das Betriebssystem oder die Speicherverwaltung und alle anderen Routinen, die sich mit der Veränderung und Aktualisierung von Datenbeständen befassen.

Bildnachweise
1457: Robyn Beeche
1458, 1459, 1464, 1480, 1481, U3: Kevin Jones
1460: Mike Clowes
1465: Liz Heaney
1470, 1471: Chris Stevens
1472: Geoff Rowley
1476: Liz Dixon
1483, 1484: Pilot Software, Liz Heaney

Fachwörter von A bis Z

Information storage and retrieval = Speicherung und Wiedergewinnung von Informationen
„Storage and Retrieval" ist ein fester Begriff für die Handhabung von Daten in Informations- (speziell Datenbank-) Systemen. Die Information kann in Form von Programm- und Datenfiles oder als Grafik vorliegen. Als Datenträger kommen nur nichtflüchtige (das heißt gegen Stromabschaltung unempfindliche) Speicher in Frage, und selbstverständlich muß der Rechner geeignete Schnittstellen für die Datenübertragung aufweisen. Softwareseitig kann das selektive Suchen anhand unterschiedlicher Kriterien durch ausgetüftelte Retrieval-Programme unterstützt werden.

Information Technology = Informationstechnologie
Informationstechnologie gibt es einerseits schon seit Tausenden von Jahren: Die Hieroglyphen, der Abakus, der Federkiel und die Druckerpresse sind Beispiele dafür. Die Entwicklung der Microelektronik und ihre Verbreitung bei rapide sinkenden Preisen im letzten Jahrzehnt hat dazu geführt, daß man bei Informationstechnologie heute primär an die elektronische Informationsdarstellung, Übertragung und Verarbeitung denkt, insbesondere für Computer, Videosysteme und Telekommunikation. Die Mehrzahl der großen Unternehmen ist heute mit Rechnern ausgerüstet. Die weltweite Kommunikation erfolgt über Satelliten wie COMSAT, und militärische Systeme halten ständig jeden Quadratmeter der Erdoberfläche unter Beobachtung. Heute werden gewaltige Forschungsanstrengungen und Geldmittel in die Informationstechnologie gesteckt.

Bildnachweise
1488, 1494, 1495, 1496, 1501, 1511: Ian McKinnell
1489: Marcus Wilson-Smith
1491: Kevin Jones
1499: Chris Stevens
1500: Artificial Intelligenz Ltd.
1502, 1503, 1505: Liz Dixon
1509: Kevin Jones, Liz Heaney
U3: Steve Cross

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Information Theory = Informationstheorie
Die Informationstheorie (begründet um 1948 von Claude Shannon am Bell-Laboratorium in New Jersey/USA) beschäftigt sich mit dem Informationsgehalt und der Übertragung von Nachrichten. Das schließt auch die Analyse der Natur von Informationen und der Eigenschaften von Nachrichtenkanälen ein.

Zur Informationstheorie gehört auch die „Codiertheorie", die sich mit der Verschlüsselung (Codierung) von Daten befaßt und auch mit der Umsetzung in „redundante Codes", die eine Übertragung von Nachrichten ohne Verlust an Informationsgehalt sicherstellen.

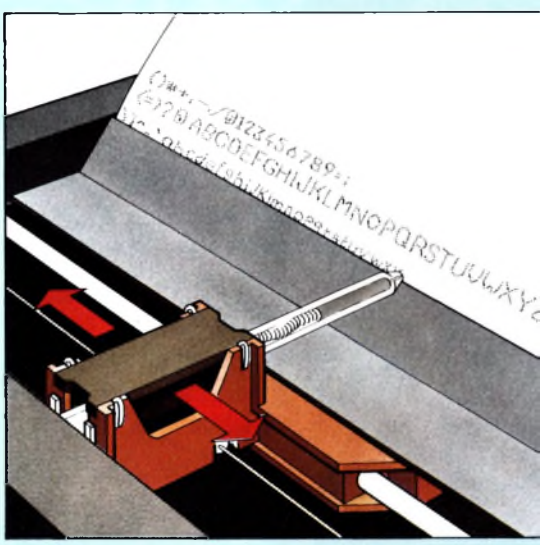
Initialisation = Initialisierung
Unter Initialisierung sind in erster Linie die Vorgänge zu verstehen, die nach dem Einschalten des Rechners ablaufen müssen, um die Betriebsbereitschaft zu gewährleisten. Dafür sind ROM-gespeicherte Routinen zuständig, die bestimmte Standardwerte in Register und diverse Speicherplätze eintragen, den Stackpointer (Stapelzeiger) auf Null setzen und die verschiedenen Ein- und Ausgabegeräte startklar machen. Die oberen und unteren Grenzen des RAM-Bereichs werden festgelegt, und die Zero Page wird erstellt. Schließlich beinhaltet die Initialisierung noch den Aufruf des Begrüßungsschirms und die Einstellung der Bildschirmeditor-Variablen.

Manchmal ist auch das Formatieren von Disketten gemeint.

Ink Jet Printer = Tintenstrahldrucker
Beim Tintenstrahlverfahren wird die Schrift auf dem Papier durch Versprühen feiner Tintentröpfchen aus einer Düse erzeugt.

Zwei Bauformen sind zu unterscheiden. Der „gepulste" Ink Jet arbeitet wie ein Nadeldrucker nach der Punktmatrix-Methode: Der Tröpfchenstrahl (aus einer Düse oder mehreren übereinander) wird bei der zeilenweisen Bewegung des Druckkopfs über das Papier abwechselnd gesperrt und freigegeben. Mit mehreren Röhrchen für verschiedenfarbige Tinten sind auch Farbausdrucke möglich.

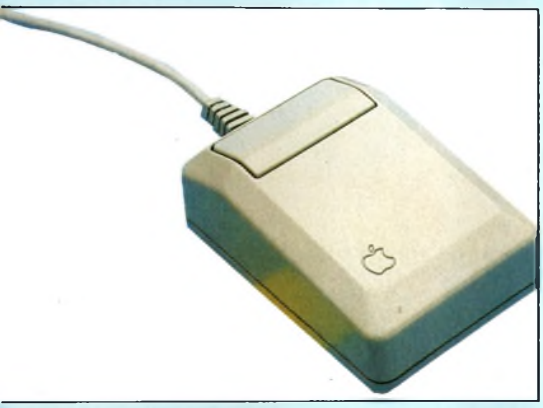
Im Gegensatz dazu ist beim „kontinuierlichen" Ink Jet ein Tröpfchenstrahl während der Erzeugung eines Zeichens ständig aktiv, wobei die (geladenen) Tröpfchen hinter der Düse durch elektrische Felder abgelenkt werden. – Jedes Zeichen wird wie mit der Feder in einem Zug geschrieben. Die Tintenstrahldrucker arbeiten lautlos mit Geschwindigkeiten bis zu einigen hundert Zeichen/s. Überdies zeichnen sie sich durch ein hervorragendes Schriftbild und hohe Variabilität aus.



Der Tintenstrahl wird beim Austritt aus der Düse in winzige elektrisch geladene Tröpfchen zerteilt, die durch ein Ablenkfeld so gesteuert werden können, wie es das jeweilige Schriftzeichen erfordert. Dabei ist ohne weiteres Schönschrift erreichbar.

Fachwörter von A bis Z

Input Device = Eingabegerät
Über Eingabegeräte wird der Rechner in Form digitaler elektrischer Signale von außen mit Information versorgt. Meist dient dazu eine Tastatur, obwohl andere Verfahren wie Spracheingabe oder Maussteuerung, auch Lichtgriffel, Grafiktabletts und natürlich Joysticks eingesetzt werden.



Die Maus ist einer von vielen Versuchen, die Eingabe benutzerfreundlicher zu gestalten. Damit soll vor allem Rechnerneulingen, die mit der Schreibmaschinentastatur und der EDV nicht vertraut sind, der Umgang mit dem Rechner erleichtert werden.

Input/Output = Ein/Ausgabe
Die Ein/Ausgabe (E/A-System) eines Computers ist für den Datenaustausch zwischen der CPU und den anderen Rechnerkomponenten zuständig. Ohne die Ein/Ausgabe hätte die CPU keine Verbindung mit der „Außenwelt". Erst über das E/t-System kann der Benutzer mit dem Prozessor kommunizieren.

Die vorrangige Aufgabe des E/A-Systems besteht darin, die von außen eintreffenden Signale in eine für die CPU verständliche Form umzusetzen. Außerdem kontrolliert das System, welches der Peripheriegeräte gerade aktiv ist und ob Daten ein- oder ausgegeben werden. Dies alles bindet eine erhebliche Prozessorkapazität, insbesondere bei der Datenausgabe über Bildschirmgeräte, deren Anzeige ja ständig aktualisiert werden muß. Deshalb werden diese Aufgaben bei den meisten Rechnern speziellen Ein/Ausgabe-Chips übertragen.

Ein vorrangiges Betätigungsfeld

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

für das E/A-System ist auch die Kommunikation mit peripheren Massenspeichern wie Cassetten und Diskettenlaufwerken.

Instruction = Befehl
Ein Befehl ist eine Handlungsanweisung für den Rechner in Form einer spezifischen Folge von Zeichen bzw. Bits. Der Befehl besteht aus dem Operationsteil (Op-Code), der die Art der Aktivität beschreibt, und dem Operanden- oder Adreßteil, der angibt, womit die Operation durchzuführen ist. In dem Befehl „LDY 8" zum Beispiel ist LDY der Op-Code und die Ziffer 8 der Operand.

Drei Gruppen von Befehlen sind zu unterscheiden: die arithmetischen Befehle wie Addition, Subtraktion, Multiplikation und Division, die logischen Befehle wie die Verknüpfungen AND/OR und die Ein/Ausgabebefehle für den Datentransfer, etwa das Laden von Registern.

Instruction Counter = Befehlszähler
Der Befehlszähler (manchmal auch „Programmzähler" oder „Befehlsadreßregister") ist ein CPU-Register, das die Speicheradresse des nächsten auszuführenden Befehls enthält. Beim Programmablauf wird der Befehlszähler schrittweise weitergeschaltet, so daß er stets den aktuellen Stand der Bearbeitung widerspiegelt.

Stößt der Rechner auf eine Verzweigung oder einen Sprungbefehl, wird der Inhalt des Befehlszählers der Anweisung entsprechend um einen festgelegten Wert erhöht oder erniedrigt. Ein Verzweigungsbefehl bewirkt, daß der aktuelle Befehlszählerstand in den Stack geschoben und die entsprechende Unterprogrammadresse geladen wird. Nach Abarbeiten der Routine setzt die CPU die Rückkehradresse vom Stack wieder in den Befehlszähler und fährt mit dem Hauptprogramm fort. Diese Vorgänge werden automatisch durchgeführt, ohne daß sich der Programmierer mit dem unteren Abläufen befassen muß.

Instruction Set = Befehlssatz
Den Vorrat an Befehlen, die ein bestimmter Prozessortyp verstehen und ausführen kann, bezeichnet man als Befehlssatz. Daraus werden alle Kommandos, zum Beispiel die BASIC-Befehle, aufgebaut. Der Befehlssatz ist prozessorintern in einem ROM festgeschrieben, Umfang und Leistungsfähigkeit sind eine Frage der CPU-Architektur. Zu dem Befehlssatz gehören unter anderem die Handhabung der Register- und Akkumulator-Inhalte, die Realisierung bedingter und unbedingter Sprünge und Verzweigungen sowie die Möglichkeit zum Abfragen und Rücksetzen der Statusregister-Flags.

Integer = Ganze Zahl
Eine Integer-Konstante oder -Variable ist immer ganzzahlig. Sie darf positiv, negativ oder Null sein. Die Zahlen 56, −43 und 0 sind Integers, 3,8 oder −2,001 oder 3 1/2 nicht.

Das Byte 1000 0001 beispielsweise kann folgende Zahlen darstellen:

Bitmuster	Interpretation	Dezimalwert
1000 0001	Integer o. Vorzeichenbit	129
1000 0001	Integer m. Vorzeichenbit	−1
1000 0001	Zweierkomplement-Integer	−127

Bildnachweis
1513: Gabrielle Izen, Holger Neuhaus
1514: Caroline Clayton
1515: David Davies
1516, 1517: Liz Heaney
1521, 1532, 1540: Ian McKinnell
1530: Kevin Jones
1538: Tony Lodge
U3: Chris Stevens

Fachwörter von A bis Z

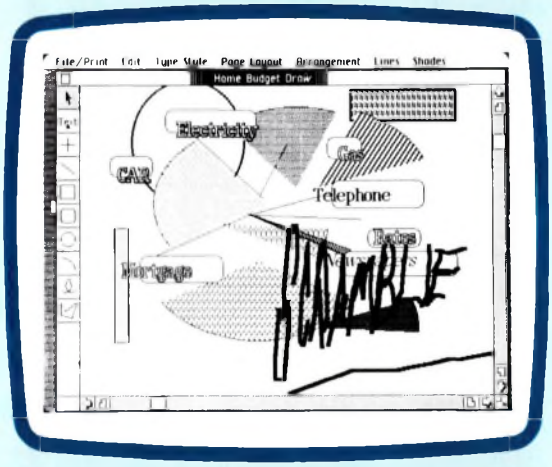
Integrated Circuit = Integrierte Schaltung
Als integrierte Schaltung oder ‚IC' wird ein elektronischer Schaltkreis bezeichnet, der in Fotoätztechnik auf einem Siliziumscheibchen aufgebaut ist. Dabei sind viele tausend Dioden- und Transistorfunktionen auf einer Fläche von wenigen Quadratmillimetern „integriert". ICs haben die gesamte Elektronikindustrie revolutioniert. Es gibt ICs für die verschiedensten Aufgaben; Hauptvorteil der Miniaturisierung gegenüber der Verwendung einzelner Bauelemente sind der geringe Leistungsbedarf und erhöhte Geschwindigkeit.

Die ersten ICs wurden Ende der fünfziger Jahre von Jack Kilby bei Texas Instruments entwickelt. Es folgten vier IC-„Generationen": Small Scale Integration (SSI), Medium Scale Integration (MSI), Large Scale Integration (LSI) und Very Large Scale Integration (VLSI).

Bei den ICs sind im wesentlichen zwei Gruppen zu unterscheiden: In Microcomputern werden bevorzugt MOS-Typen (Metall-Oxid-Semiconductor) verwendet, die sich durch hohe Packungsdichte und geringen Leistungsbedarf auszeichnen und extrem kostengünstig herzustellen sind. Bei größeren Rechnern werden dagegen vorwiegend bipolare ICs eingesetzt, bei denen die Schaltung auf dem Siliziumplättchen mit positiven und negativen Transistorelementen aufgebaut wird – dieses Verfahren ist wegen der unterschiedlichen Halbleiterdotierung aufwendiger als die MOS-Technologie. Die Packungsdichte ist erheblich geringer, der Leistungsbedarf größer; bipolare ICs sind jedoch schneller als MOS-Bausteine.

Intelligent Terminal = Intelligentes Terminal
Mit einem intelligenten Terminal lassen sich in begrenztem Umfang Daten bearbeiten – die Ergebnisse können dann an Großrechner weitergegeben werden. Diese Vorverarbeitung, zum Beispiel das Editieren, wird durch einen eigenen Prozessor möglich, der damit den Hauptrechner für andere Aufgaben freihält.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



Lisadraw ist ein interaktives Grafikprogramm-Paket. Jedes Pixel auf dem Bildschirm kann einzeln angesprochen werden, so daß sich eine hohe Grafikauflösung ergibt.

Interactive Graphics = Interaktive Grafik
Von ‚interaktiver' Grafik spricht man, wenn ein System die Erstellung und Modifizierung von Bildern und Grafik in unmittelbarem Wechselspiel zwischen Rechner und Benutzer erlaubt. Diese Technik findet bei Microcomputern zunehmend Anwendung. Bekannt ist das integrierte Software-Paket ‚Lotus 1-2-3', das die sofortige Darstellung von Tabellendaten als Kurven oder Torten- bzw. Balkendiagramm ermöglicht.

Interface = Schnittstelle
Eine Schnittstelle kann sowohl eine Anpassungsschaltung sein, die den Datenaustausch zwischen zwei Systemen ermöglicht, als auch eine spezielle Routine für die Kommunikation zwischen unterschiedlichen Programm-Modulen (‚Software-Interface'). Aufgabe der Schnittstelle ist die Umsetzung von Informationen in eine Form, die vom Empfängersystem verstanden wird.

Zu einer Hardware-Schnittstelle gehören neben der Elektronik auch Verbindungselemente wie Kabel und Stecker; ein Software-Interface muß als „Brücke" zwischen Programmteilen den Austausch von Parametern und Variablen vermitteln.

Interpreter = Interpreter
Die CPU eines Rechners verarbeitet Bitmuster nach vorgegebenen logischen Regeln – Menschen drükken ihre Gedanken in Begriffen, Wörtern und Symbolen aus. Diesen Unterschied sollen Programmiersprachen lösen, indem sie einerseits dem Benutzer entgegenkommen und andererseits für den Rechner unter Zwischenschaltung eines Übersetzungsprogramms verständlich sind.

Bei der Eingabe eines Programms wandelt der Interpreter die einzelnen Anweisungen in eine kompakte Kommandoform um. Beim Programmlauf werden die Kommandos dann vom Interpreter in Maschinenbefehle umgesetzt und unmittelbar ausgeführt, ohne daß der erzeugte Code gespeichert wird.

Diese Arbeitsweise macht den Interpreter besonders geeignet für die Entwicklung von Programmen und für Testläufe. Sie sorgt allerdings, etwa beim BASIC-Interpreter, für die weithin bekannte Langsamkeit dieser Programmiersprache.

Bildnachweise
1541: Helen Zahorodnyj
1542: Steve Cross
1543: Courtesy of U.I.P.
1544, 1555, 1562 Ian McKinnell
1549: Kevin Jones
1557: Caroline Clayton
1559: Jojo
1560: Timothy Ginn
1564: Gabrielle Izen, Hotte Wurst
1565: Liz Dixon

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Interrupt = Unterbrechung
Es gibt Software- und Hardware-Interrupts. Software-Unterbrechungen werden innerhalb eines Programms erzeugt, und ihre Behandlung übernimmt das Betriebssystem. Hardwaremäßig erfolgt die Unterbrechung durch einen Impuls am Interrupt-Eingang des Prozessors, was den Programmabbruch und meist ein Reset bewirkt, das das System in den Einschaltzustand versetzt.

Interrupts werden für die verschiedensten Zwecke gebraucht. Wenn ein Programm für eine bestimmte Aufgabe die Hilfe des Betriebssystems benötigt, schickt es eine „freiwillige" Interrupt-Anforderung an die Systemsteuerung, die daraufhin entsprechend reagiert.

Etwas ähnliches geschieht bei der Auslösung eines „unfreiwilligen" Interrupts, wenn etwa das ROM-gespeicherte BASIC in einem Programm Fehler entdeckt. Die Unterbrechung führt dann zu einem Programmabsturz und zur Ausgabe der zugehörigen Fehlermeldung. „Timer"-Interrupts treten in festen Zeitintervallen auf, wenn die Systemuhr weitergestellt wird. Sie können als Zeitmarken zur Überwachung von Vorgängen verwendet werden.

Job Control Language = Kommandosprache
Sie umfaßt die Anweisungen, mit denen das Betriebssystem für die selbständige Bearbeitung von Benutzeraufträgen programmiert wird. Bei Microcomputern ist so etwas noch selten, obwohl die Stapelverarbeitung unter CP/M schon einen Anfang darstellt. Dagegen bedienen sich mittlere Rechner und Großanlagen verbreitet der JCL und arbeiten damit kontinuierlich die verschiedenen Benutzerprogramme ab, die als Job-Kette auf Band oder Platte gespeichert werden. Meist benötigt ein Job mehrere Systemprogramme nacheinander, so etwa zum Drucken, Lesen, Sortieren usw. Statt diese Routinen einzeln durch den Operateur zu aktivieren, wird ein JCL-Programm zusammengestellt, das Laden und Starten der Systemroutinen übernimmt.

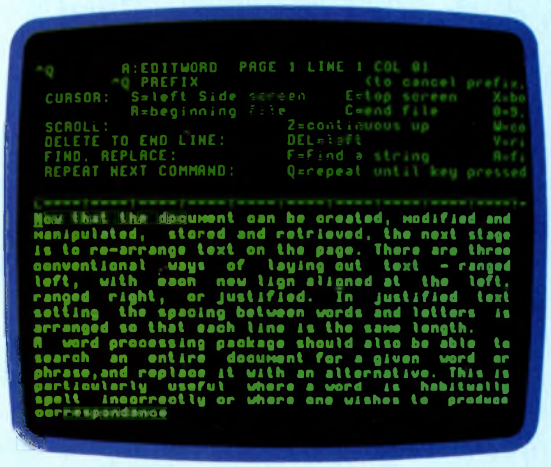
Jump = Sprung
Ein Sprungbefehl bewirkt, daß der Rechner aus der laufenden Anweisungsfolge aussteigt und das Programm an anderer Stelle fortsetzt. In BASIC lautet der Sprungbefehl „GOTO". Er hat im Unterschied zur GOSUB-Anweisung nicht den Nebeneffekt, daß für den späteren Rücksprung der aktuelle Befehlszählerstand auf den Stack geschoben wird. Die Rückkehr zum Ausgangspunkt kann daher nur über ein neuerliches GOTO erfolgen.

Ein „bedingter" Sprung wird mit IF...THEN (Wenn... dann) eingeleitet und nur dann ausgeführt, wenn eine bestimmte Bedingung erfüllt ist: IF Bedingung erfüllt THEN Sprung.

Junction = Übergang
Die Grenzzone zwischen Halbleitern mit unterschiedlichen elektrischen Eigenschaften (auch zwischen Metall und Halbleiter) wird als Übergang bezeichnet, sie ist entscheidend für das Verhalten von Bauteilen. Bei einem p-n-Übergang beispielsweise stößt ein p-dotierter (d. h. positive Ladungsträger freisetzender) Halbleiter an einen n-dotierten Bereich (mit negativen Ladungsträgern), wobei sich durch Ladungsträgeraustausch und Rekombination eine elektrische „Sperrschicht" und gleichzeitig eine Spannung zwischen p- und n-Material ausbildet. Durch Anlegen einer äußeren Spannung kann die Sperrschicht mehr oder weniger abgebaut und damit der Stromfluß durch den Übergang gesteuert werden. Darauf beruht die Gleichrichterwirkung von Dioden und im Prinzip auch der Verstärkungsmechanismus bei Transistoren.

Justification = Justierung
Das Justieren von Texten beim Blocksatz beherrscht fast jedes Textverarbeitungsprogramm: Die Buchstaben werden innerhalb der Zeile so ausgerichtet, daß sie am linken und am rechten Rand bündig stehen.

Bei der Maschinencode-Programmierung versteht man unter Justification die Verschiebung von Registerinhalten mit dem Ziel, die erste oder letzte von Null verschiedene Stelle in die höchst- bzw. niedrigstwertige Bitposition zu bringen. Das ist für Abfragezwecke, insbesondere bei der „Normalisierung" im Gleitkommaformat von Bedeutung.



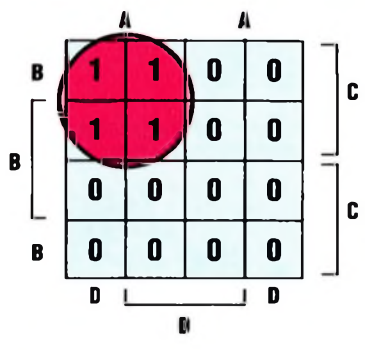
Das Justieren der Zeilenränder ist eigentlich bei jedem Textverarbeitungsprogramm vorgesehen. Wenn Sie das Bildschirmfoto genau betrachten, sehen Sie, daß der bündige Zeilenabschluß durch Variation der Leerraumlängen zwischen den Wörtern erreicht wird.

Bildnachweise
1569, 1570: Science Photo Library LTD
1570, 1571, 1582, 1583, 1587: Kevin Jones
1572, 1573: Uta Brandl
1574, 1575: Chris Stevens
1576, 1577, 1586: Caroline Clayton
1583, 1584, 1596, U3: Ian McKinnell
1589: Liz Dixon
1591: Tony Sleep
1592: Chris Barker
1593: Europian Marketing Division

Fachwörter von A bis Z

K = K
Das kleine „k" ist die Abkürzung für die Vorsilbe „Kilo", das heißt, den Faktor 1000 bei physikalisch-technischen Einheiten. In der Datenverarbeitung bezeichnet die Vorsilbe Kilo dagegen den Faktor 2^{10}=1024 (mit einem großen „K" abgekürzt). 1 Kilobyte sind 1024 Byte, ünd 64 KByte sind 64 × 1024=65 536 Byte. Bei Speicherkapazitäts-Angaben wird die Einheit Byte auch oft weggelassen – man spricht einfach von „64 K RAM".

Karnaugh Map = Karnaugh-Diagramm
Ein Karnaugh-Diagramm ist die grafische Darstellung einer Wahrheitstabelle (zweidimensional). So wie ein Venn-Diagramm Durchschnitt und Vereinigung von Mengen deutlich macht, illustriert das Karnaugh-Diagramm logische Verknüpfungen.



Komplizierte logische Ausdrücke lassen sich durch Umformung nach den Regeln der Booleschen Algebra vereinfachen, was aber mühsam und fehlerträchtig ist. Ein Karnaugh-Diagramm liefert dagegen ein unmittelbares Abbild der Verknüpfungen und erlaubt die Vereinfachung aus der Anschauung heraus. Nebenstehend ist der Ausdruck A AND B dargestellt.

Kernel = Kern
Der Kern eines Rechner-Betriebssystems beinhaltet die lebenswichtigen Funktionen. Wenn dieser nicht geladen ist oder nicht richtig arbeitet, läuft das System nicht; er stellt das Hauptsteuerprogramm dar. Der Kern sorgt durch unmittelbare Anweisungen an die CPU für die Aktivierung und Überwachung aller anderen Systemprogramme, die beim Rechnerbetrieb benutzt werden.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Key = Taste
Das englische Wort „Key" ist in der Datenverarbeitung mehrdeutig: Wenn damit nicht die einzelne Taste eines Keyboard gemeint ist, dann ein „Schlüssel" im übertragenen Sinn, der den Zugang zu Informationen eröffnet.

Bei einer Datenbank bezeichnet man z. B. das Schlüsselfeld für Suchvorgänge als „Key". Die Datensätze bilden jeweils eine aus mehreren Feldern bestehende Einheit. In einem Adressenverzeichnis etwa bilden Name, Anschrift und Rufnummer jedes Teilnehmers den individuellen Datensatz – es gibt ein Namen-, Adressen- und Nummernfeld. Wird ein Auszug aus dem Verzeichnis benötigt, ist eins dieser Felder als „Key" für die Auswahl anzugeben.

Bei einigen Datenbanksystemen ist die Suche unter mehreren Gesichtspunkten möglich; dabei können Sie dann als Hauptkriterium einen „Primärschlüssel" (Primary Key) und dazu einen „Sekundärschlüssel" (Secondary Key) benennen usw. Kompliziert wird es, wenn das System in einem der Felder die Eingabe von unformatierten Textbeschreibungen in Form ganzer Sätze oder sogar Abschnitte vorsieht.

Bei geschützten Dateien dient oft ein Schlüsselwort (Paßwort) oder eine Schlüsselnummer zur Benutzeridentifikation – damit wird die Sperre gegen möglichen unbefugten Gebrauch geöffnet.

Schließlich bezeichnet man auch den Umsetzungscode, nach dem bei Chiffrierverfahren die Nachrichten verschlüsselt werden, als Key. Bei einem Substitutions-Code enthält die Key-Tabelle die Ersatzzeichen für die Originalbuchstaben. Dabei wird die Zuordnung mit Hilfe von Zufallszahlengeneratoren im Rechner erzeugt, so daß der Code ohne den zugehörigen Schlüssel praktisch nicht zu knacken ist.

Keypad = Tastenblock
Ein „Keypad" ist ein kleines Tastenfeld für spezielle Zwecke, zum Beispiel für die Eingabe von Zahlenwerten. Bei vielen Microcomputern enthält die Tastatur einen numerischen Block, in dem Ziffern- und Funktionstasten nach dem gleichen Schema wie bei Taschen- oder Tischrechnern angeordnet sind.

Bei manchen Rechnern ist der numerische Block in das alphanumerische Tastenfeld integriert, indem bestimmten Tasten eine Doppelfunktion zugewiesen ist. Meist werden die Buchstaben M, J, K, L, U, I und O zusätzlich mit den Ziffern 0–6 belegt; zusammen mit den Ziffern 7, 8 und 9, die in der obersten Tastenreihe darüberliegen.

Microrechner wie der ACT Apricot oder der IBM PC haben dagegen einen getrennten Ziffernblock, der rechts neben dem alphanumerischen Feld liegt.

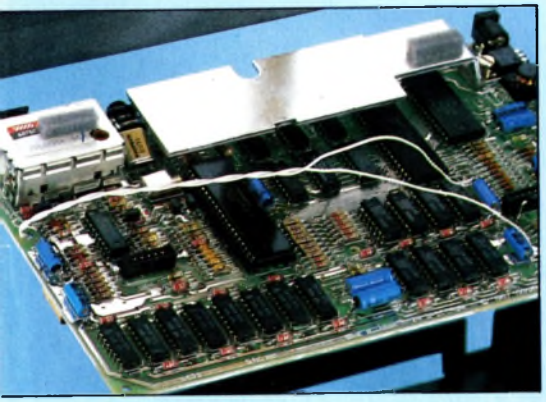
Bildnachweise
1597: Science Photo Library
1598: Courtesy of Robocom
1599: Modelbox des Theatre Royal, Stratford East; Courtesy of Whitechapel Computer Centre
1602, 1603: Ian McKinnell
1605: Liz Dixon
1606, 1607: Ian McKinnell
1609–1611: Crispin Thomas
1612: Terry Kennett
1613: Ian McKinnell; Allan Martin; Courtesy of Simulation Database Project, Glasgow College of Education
1615: David Higham
1618: Liz Dixon
1620: Ian McKinnell
1621: Liz Dixon
1622–1624: Kevin Jones

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Keypunch = Locher
Lochkarten oder -streifen gehören zu den klassischen Datenträgern. Bei Karten werden Buchstaben und Ziffern als Lochmuster auf einem Hartpapierblatt mit zwölf Zeilen à 80 Spalten dargestellt. Dieses Format war als „Hollerith-Code" schon vor der Jahrhundertwende bei Tabelliermaschinen – den ersten Bürorechnern – gebräuchlich.

Keyword = Schlüsselwort
Um das Programmieren benutzerfreundlicher zu machen, arbeiten alle höheren Programmiersprachen mit sogenannten „Schlüsselwörtern" – das sind Vokabeln, die mit genau definierter Bedeutung wesentliche Strukturelemente der Sprache darstellen. Wenn der Compiler oder Interpreter auf ein solches „Keyword" trifft, verfährt er nach einer bestimmten vorgegebenen Routine. Auch der BASIC-Befehlsvorrat ist aus derartigen Schlüsselwörtern aufgebaut: PRINT, LET, GOTO, IF...THEN usw. haben festgelegte Funktionen und dürfen deshalb nicht als Variablennamen verwendet werden.



Die beiden Leitungen, die hier über der Leiterplatte des Sinclair Spectrum liegen, führen zu einer separaten Reset-Taste und verwandeln den Spectrum in einen Spectrum+. Die (kostspielige) Alternative für diesen typischen „Kludge" hätte darin bestanden, eine neue Platine zu entwerfen, um die zusätzlichen Leiterbahnen unterzubringen.

Kludge = Zwischenlösung
Sind bei einem Gerät während des Fertigungsanlaufs plötzlich wegen irgendwelcher Probleme noch Änderungen erforderlich, hilft nur Improvisation. Die „Zwischenlösung", die dann schnellstens aus vorhandenen Teilen zusammengebaut wird, bezeichnen die Engländer als „Kludge" Bei den ersten QL-Modellen beispielsweise konnte das Betriebssystem nicht in den vorgesehenen ROMs untergebracht werden. Als Zwischenlösung wurde ein zusätzliches EPROM eingebaut.

Label = Markierung
„Label" sind Kennzeichnungen zur Identifizierung oder Abgrenzung von Informationen in der Datenverarbeitung. Bei der Assembler-Programmierung zum Beispiel werden Label zur Markierung von Anweisungen und als symbolische Adressen benutzt, etwa in folgender Form:

Label-Feld	Anweisungs-Feld	Operanden-Feld
LOOP	LDA	TEST
EXIT	BNE	LOOP

LOOP, TEST und EXIT sind hier Label, die bei der Übersetzung des Programms in den Maschinencode durch die entsprechenden Adressen ersetzt werden.

Eine andere Label-Art sind kommentierende Zusätze im Programm, zum Beispiel REMarks in BASIC, aus denen hervorgeht, welche Aufgabe der folgende Block hat. In dieser Form sind Label wesentliche Elemente der strukturierten Programmierung, die dem Programmierer die Übersicht erleichtern.

Die strukturierte Programmierung macht auch von „Prozeduren" Gebrauch. Das sind Unterprogramme, auf die mit Hilfe eines Label Bezug genommen werden kann.

Ein Label kann auch eine kurze Information auf Cassette oder Diskette zur Charakterisierung von Merkmalen der gespeicherten Dateien sein. Ein „Volume Label" beschreibt den gesamten Datenträger, während ein „Header Label" oder Dateikopfsatz nur für eine einzelne Datei zuständig ist.

Schließlich gibt es bei komfortablen Tabellenkalkulationsprogrammen noch die Möglichkeit, Gruppen von verwandten Tabellenfeldern durch Label zu markieren und dann Rechnungen mit sämtlichen Feldern auf einmal durchzuführen, indem die Label in eine Formel eingesetzt werden. Sind etwa die Verkaufsziffern für ein Produkt ein Jahr lang notiert worden, läßt sich daraus eine Verkaufsprognose für das nächste Jahr erstellen, indem man bei den Vorjahreszahlen prozentuale Auf- und Abschläge berücksichtigt. Ohne die Verwendung von Labels müßte man für jedes betroffene Tabellenfeld eine Gleichung der Art
A4 × 110% = B4
ansetzen, die Beziehung also viele Male hinschreiben. Wenn Label zugelassen sind, muß man stattdessen nur einmal die Anweisung geben
Alter Umsatz × 110% =
Neuer Umsatz
und die zugehörigen Feldbereiche entsprechend markieren. Die Formel spricht dann alle Felder mit dem Label „Alter Umsatz" an und überträgt die Ergebnisse in den Datensatz mit dem Label „Neuer Umsatz". Dieses Verfahren erleichtert den Umgang mit Tabellensystemen in erheblichem Maße.

Bildnachweise
1625: Johannes Heidenhain
1626, 1627: Kevin Jones
1630, 1637: Ian McKinnell
1638, 1639: Chris Stevens
1641: Tony Sleep
1642: Kevin Jones
1643: Ian McKinnell
1644, 1645: Kevin Jones
1646: David Higham, Kevin Jones
1650: Caroline Clayton
U3: Chris Stevens

Fachwörter von A bis Z

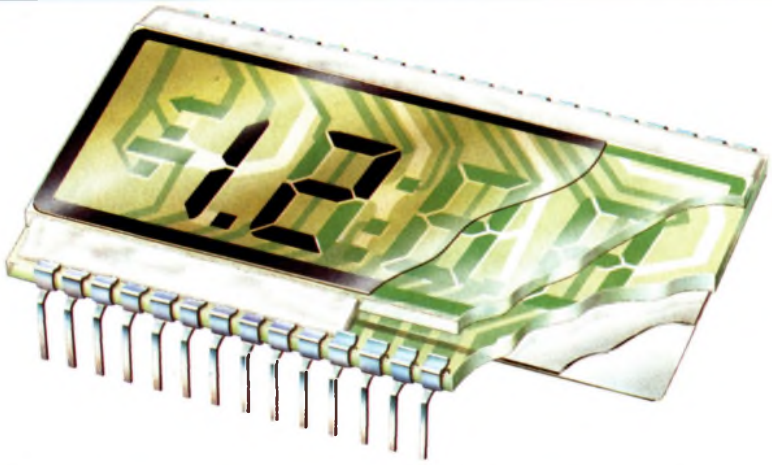
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Language Construct = Sprachkonstrukt
Jede Programmiersprache verfügt über eine Reihe von Strukturelementen zur Realisierung häufig benötigter Funktionen. Dazu gehören „Schlüsselwörter" wie PRINT für die Datenausgabe über den Bildschirm.

Eine Programmiersprache muß aber auch kompliziertere Abläufe unterstützen können, die eine Folge von Einzelaktionen erfordern. Die entsprechenden Strukturen bezeichnet man als „Sprachkonstrukte". Ein Beispiel ist die Schleifenanweisung FOR . . . NEXT. Was innerhalb der Schleife geschieht, bestimmt dabei der Programmierer, aber der Aufbau der Schleife ist fest definiert. Konstrukte für Wertzuweisungen, Entscheidungen, Adressierung und Ein/Ausgabe sind wesentliche Elemente jeder Programmiersprache.

Laser Printer = Laserdrucker
Ein Laserdrucker ist ein hochwertiges Ausgabegerät, das hohe Druckgeschwindigkeiten erreicht. Die Zeichen werden als dichtes Punktmuster mit einem Laserstrahl zunächst auf die elektrisch geladene Oberfläche eines Fotohalbleiters geschrieben. Wo Licht auffällt, lädt sich der Halbleiter um; es entsteht ein „Ladungsbild". Dieses wird mit einem Tonerpulver „entwickelt", das an den belichteten Stellen haftenbleibt (ähnlich wie bei vielen Fotokopierern), und dann auf Papier übertragen und eingebrannt. Laserdrucker sind farb- und grafiktauglich und können beliebige Schriftarten erzeugen.

LCD = Flüssigkristallanzeige
LCDs (Liquid Crystal Displays) werden als Bildschirmersatz für Microcomputer eingesetzt. Alphanumerische Anzeigen sind als Siebensegment-Felder aufgebaut. Sie bestehen aus einer Sandwich-Anordnung von Elektrodenplatten und Polarisationsfiltern mit einer „kristallinen" Flüssigkeit zwischen den einzelnen Elementen und einer reflektierenden Unterlage. Bei angelegter Spannung zwischen den Elektroden wird die zuvor transparente Flüssigkeit lichtundurchlässig, und das betreffende Segment erscheint dunkel. Das Hauptargument für die Verwendung von LCDs, vor allem bei tragbaren Computern, sind ihr geringer Platz- und Leistungsbedarf. Ein wesentlicher Nachteil ist die große Trägheit: Die Anzeige kann mit dem Tempo einer geübten Schreibkraft nicht schritthalten.



Ein LCD enthält zwischen zwei Polarisationsfiltern eine dünne Flüssigkristallschicht. Die Glasplatten tragen an der Innenseite ein Elektrodenmuster, das den Zeichenkonturen entspricht. Ist Spannung an die Elektroden gelegt, werden die Flüssigkeitsmoleküle durch das elektrische Feld ausgerichtet. Dabei ändert sich die Transparenz, und die Zeichensegmente heben sich gegen den Hintergrund ab.

LED = Leuchtdiode
LED steht für „Light Emitting Diode", das heißt für eine Halbleiterdiode, die bei Stromdurchgang Licht emittiert. Einzelne LEDs werden bei Computern oft für Kontrolleuchten eingesetzt; außerdem lassen sich LEDs auch zu 7-Segment-Blöcken zusammensetzen und wie LCDs als alphanumerische Anzeigen verwenden. Davon wurde früher bei Taschenrechnern viel Gebrauch gemacht – in diesem Bereich sind die LEDs wegen ihres hohen Leistungsbedarfs heute weitgehend von den LCDs verdrängt worden.

Lexical Analysis = Lexikalische Analyse
Weil die Verarbeitung von Programmen im Rechner nach strengen Regeln erfolgt, muß der eingegebene Programmtext vor dem eigentlichen Übersetzen in eine standardisierte Form gebracht werden. Das besorgt ein Teil des Compilers mit einem Satz spezieller Routinen durch die „lexikalische Analyse". Die Quellenanweisungen werden in ihre Bestandteile zerlegt, Zahlenkonstanten in die interne Darstellung gebracht, Variablennamen zusammengestellt.

Bildnachweise
1653: Marcus Wilson Smith
1655, 1671: Carolyne Clayton
1658–1660, 1666, 1667: Kevin Jones
1664, 1665, 1668, 1669, 1677: Ian McKinnell
1672, 1673: Liz Dixon
1675: Steve Cross
U4: Tony Sleep

Fachwörter von A bis Z

Light Pen = Lichtgriffel
Der Lichtgriffel erlaubt das Arbeiten am Bildschirm ohne Tastaturbenutzung. Sie setzen einfach die Griffelspitze auf den Schirm und können damit Rechnerfunktionen auslösen, Menüvarianten auswählen oder Linien ziehen.

Das praktische Gerät ist über ein Kabel mit dem Rechner verbunden. In die Spitze sind eine Linse und ein Fotodetektor eingebaut, der bei Lichteinfall ein elektrisches Signal abgibt. Es wird verstärkt, zu einem definierten Impuls geformt und an den Rechner weitergeleitet.

Seine Funktion beruht darauf, daß der Zeitpunkt registriert wird, zu dem das Lichtsignal beim zeilenweisen Aufbau des Bildes die Stiftspitze passiert. Der Video-Chip des Rechners „weiß", welche Zeile gerade geschrieben wird und wann sie begonnen worden ist. Aus der Zeit, die bis zum Eintreffen des Griffelsignals vergeht, kann der Computer anhand der bekannten Geschwindigkeit des Bildaufbaus ausrechnen, wo die Spitze auf den Schirm gesetzt wurde.

Der Lichtgriffel enthält selbst keine Lichtquelle, sondern er registriert den Vorbeigang des Kathodenstrahls auf dem Bildschirm; daraus erkennt der Rechner die Stelle, auf die mit der Stiftspitze gezeigt wurde. Der Lichtgriffel eignet sich vor allem für die interaktive Grafik und für die Menüsteuerung.

Line Printer = Zeilendrucker
Ein Zeilendrucker ist ein extrem schneller, mechanischer Drucker. Zeilendrucker wurden früher hauptsächlich in Verbindung mit Großrechnern eingesetzt; sie sind heute in der Druckgeschwindigkeit (etwa 30 Zeilen/s) von den elektrostatischen Laserdruckern überholt worden. Die englische Bezeichnung „Line Printer" wird auch für Typenrad- und Matrixdrucker verwendet.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Unabhängig von der technischen Ausführung haben alle Drucker gemein, daß die Daten vom Rechner zunächst in einem RAM-Buffer abgelegt werden. Die Druckgeschwindigkeit ist in jedem Fall kleiner als die Übertragungsrate, so daß die CPU ohne Buffer für andere Funktionen blockiert wäre. So muß sie nur ab und zu einen Bufferinhalt abschicken und kann zwischendurch anderweitig tätig sein – der Drucker meldet sich, wenn er den Buffer abgearbeitet hat und fordert neue Daten an.

LISP = LISP
LISP ist eine Programmiersprache, die speziell für die Verarbeitung von Informationslisten (LISt Processing) ausgelegt ist. Die Listen werden mit Unterstützung des Systems definiert und mit Namen versehen. Mit den Listenelementen kann beliebig hantiert werden, etwa um festzustellen, ob bestimmte Bedingungen erfüllt sind oder nicht.

Die Programmierung in LISP besteht in der Festlegung der Verarbeitungsfunktionen; LISP ist eine „funktionale" Sprache, wobei ähnlich wie in LOGO komplexe Anweisungen aufgebaut werden können.

Local Area Network = Lokalnetz
Ein Lokalnetz oder (engl. abgek.) „LAN" besteht aus einer Anzahl von Rechnern, die untereinander und mit einer gemeinsamen Peripherie kommunizieren. Dabei ist weder ein einheitlicher Rechnertyp noch die enge räumliche Nachbarschaft der Computer zwingend.

Ein LAN kann auf dreierlei Weise realisiert werden: Beim Stern-System ist jeder beteiligte Computer mit einem Zentralrechner verbunden, der das ganze Netz und die Peripherie steuert. Der Datenverkehr zwischen zwei Satellit-Rechnern erfolgt dabei grundsätzlich über die Zentrale.

Beim Ringnetz sind die Rechner dagegen alle hintereinander geschaltet. Ringnetze sind für größere Teilnehmerzahlen ungeeignet, weil die Information dann auf dem Weg ans Ziel über zu viele Rechner laufen muß.

Beim Bus-System als drittem Typ, zum Beispiel beim Acorn-B-Econet, wird jeder Rechner über ein eigenes Anschlußkabel mit einer gemeinsamen Bus-Leitung verbunden, so daß unmittelbare Kontaktaufnahme zwischen Teilnehmern möglich ist, ohne daß eine Zwischenstation in Anspruch genommen werden muß.

Logic = Logik
Die Logik ist das Fundament der gesamten Datenverarbeitung. Die Logik als Wissenschaft befaßt sich mit den Formalismen, die durch Verknüpfung einer Anzahl von Parametern – etwa mit Hilfe der Booleschen Algebra – zu bestimmten Schlußfolgerungen führen. Logische Voraussetzungen und Aussagen sind entweder wahr oder falsch – Zwischenwerte gibt es nicht. Deshalb eignet sich auch das Zweiersystem, mit dem jeder Digitalrechner arbeitet, ganz besonders für die Darstellung logischer Zusammenhänge.

Als Logik wird oft auch die Elektronik für die Ausführung der zugehörigen Operationen bezeichnet. Im Rechner finden Sie zur Nachbildung der logischen Verknüpfungen AND-, NAND-, NOT-, OR-, NOR- und EXOR-Schaltkreise, die die Basis des ganzen digitalen Systems bilden.

Bildnachweise
1681, 1686, 1687, 1692: Ian McKinnell
1689, 1705, 1706: Liz Dixon
1691: Tony Sleep
1694, 1695: Kevin Jones
1696: Caroline Clayton
1701: Alan Adler

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

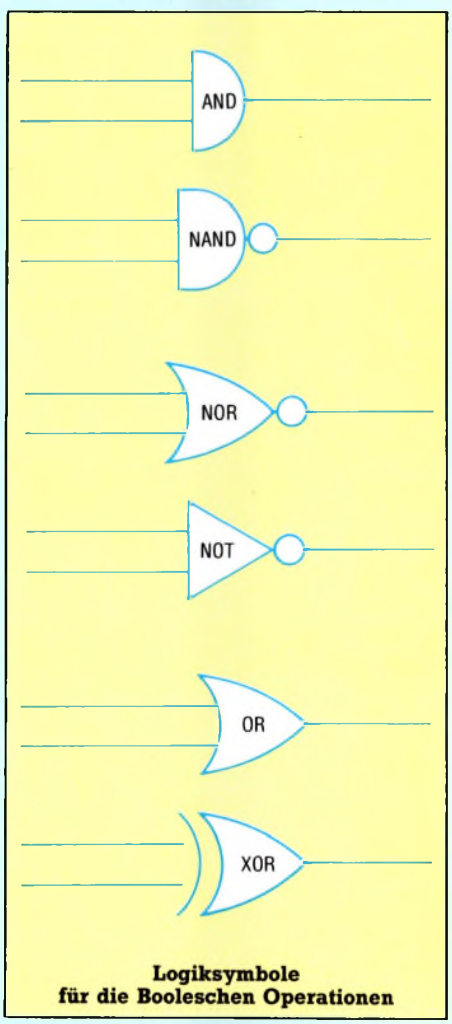
Logic Gate = Logikgatter
Schaltkreise zur Realisierung logischer Verknüpfungen heißen Logik-Gatter. Ein Gatter hat ein oder zwei Ausgänge und bis zu acht Eingänge; je nach den Eingangsspannungspegeln und der logischen Funktion des Gatters werden die Ausgänge auf „High" (Betriebsspannung) oder „Low" (Nullpegel) gesetzt.

Die Arbeitsweise eines Gatters läßt sich mit einer Anzahl Schalter vergleichen, die in einer bestimmten Anordnung betätigt werden; ein ODER-Gatter entspricht beispielsweise zwei parallel liegenden Schaltern. Als elektronische Schalter werden dabei bipolare sowie Feldeffekt-Transistoren verwendet. Ein Transistor hat drei Anschlüsse oder Elektroden, sie heißen beim bipolaren Transistor Emitter, Kollektor und Basis. Die Basis ist die Steuerelektrode; das dort anliegende Signal bestimmt, ob zwischen Emitter und Kollektor eine leitende Verbindung besteht oder ob der Transistor gesperrt ist. Die Gattereingänge sind jeweils mit Basis-Elektroden verbunden, und die Emitter-Kollektor-Strecken schalten definierte Spannungspegel auf die Ausgänge durch. Die Gattertypen – AND/OR/NAND/NOR/NOT/EXOR – unterscheiden sich in der Anordnung der Schalttransistoren.

Logic State = Logischer Zustand
Den Null- und Eins-Binärwerten lassen sich die logischen Zustände „wahr" und „falsch" zuordnen.

Der zugehörige Spannungspegel hängt dagegen von der verwendeten „Logikfamilie" ab. Bei einem Gatter, das beispielsweise mit +5V Betriebsspannung arbeitet, entsprechen diese 5V bei positiver Logik dem Zustand „wahr", 0 V bedeutet dementsprechend „falsch".

Logic Symbols = Logiksymbole
Jeglicher Systementwurf – ob nun in Form eines Flußdiagramms oder einer elektronischen Schaltung – sollte sich möglichst auf einen Satz von grafischen Symbolen stützen, mit dem die verschiedenen Komponenten in eindeutiger und allgemeinverständlicher Weise zu kennzeichnen sind.



Logiksymbole für die Booleschen Operationen

Für den Entwurf logischer Schaltkreise gibt es international genormte Logiksymbole. Die Zeichen für die gebräuchlichsten Verknüpfungen der Booleschen Algebra sind in der Abbildung wiedergegeben; neben diesen Symbolen werden noch genormte Sinnbilder für die verschiedenen Flipflop-Typen verwendet.

Logging On = Einloggen
Beim Rechnerbetrieb ist es oft unerläßlich, den Zugriff unbefugter Personen auf die Anlage insgesamt oder auf bestimmte Dateien zuverlässig zu verhindern. Daher muß sich der Benutzer bei größeren Systemen meist erst einmal „einloggen".

Zum Einloggen gehört normalerweise ein Paßwort. Wenn viele Leute das gleiche Terminal benutzen, wird häufig ein gemeinsames Paßwort als generelle Zugangsberechtigung verwendet und dazu noch ein persönliches, das die eigenen Dateien oder andere Systembereiche freigibt.

Nach Beendigung seiner Tätigkeit muß sich der Benutzer über eine „Log off"-Routine abmelden (ausloggen). Damit ist die Verbindung zwischen Terminal und Rechner unterbrochen. Bei Großanlagen wird der Benutzer im allgemeinen automatisch abgehängt, wenn länger keine Eingabe erfolgt.

Vor allem mit der wachsenden Verbreitung von Computernetzen in den letzten Jahren hat auch die Zahl der Hacker zugenommen, die Paßworte knacken und unberechtigt in fremde Systeme eindringen. Zumindest für Telefon-Datennetze gibt es jedoch neuerdings eine wirkungsvolle Methode, die den Hackern kaum noch Chancen läßt: Der Rechner unterbricht die Verbindung, nachdem das Paßwort eingegeben ist, und ruft selbst den rechtmäßigen Besitzer an.

Bildnachweise
1709: Nick Harris
1710: Liz Heany
1711: Tony Sleep
1712, 1721, U3: Liz Dixon
1714, 1724: Kevin Jones
1717: Sperry Ltd.
1722: Raster Technologies
1723: G. Entis, R. Chuang
1725, 1728, 1729: Ian McKinnell
1731–1733: Caroline Clayton
1734: Sporting Pictures
1734–1736: Marcus Wilson-Smith

Fachwörter von A bis Z

Loop = Schleife
Unter einer Schleife ist ein Programmabschnitt zu verstehen, der durch wiederholten Rücksprung mehrfach ausgeführt werden kann. Es gibt „bedingte“ und „unbedingte“ Schleifen.

Bei einer bedingten Schleife wird der zyklische Programmteil nur solange durchlaufen, bis eine bestimmte Bedingung erfüllt ist. In BASIC kann dieser Schleifentyp unterschiedlich realisiert werden, etwa durch die Anweisung DO LOOP...UNTIL, WHILE...WEND, REPEAT...UNTIL und FOR...NEXT.

Bei einer unbedingten Schleife erfolgt der Rücksprung zum Eingang ohne Bedingungsabfrage; ein Ausstieg ist nur durch äußeres Eingreifen möglich. Für solche Schleifen wird in BASIC das einfache GOTO verwendet – mit der Anweisung 10 GOTO 10 beispielsweise kann man aufs bequemste eine Endlosschleife ausführen lassen.

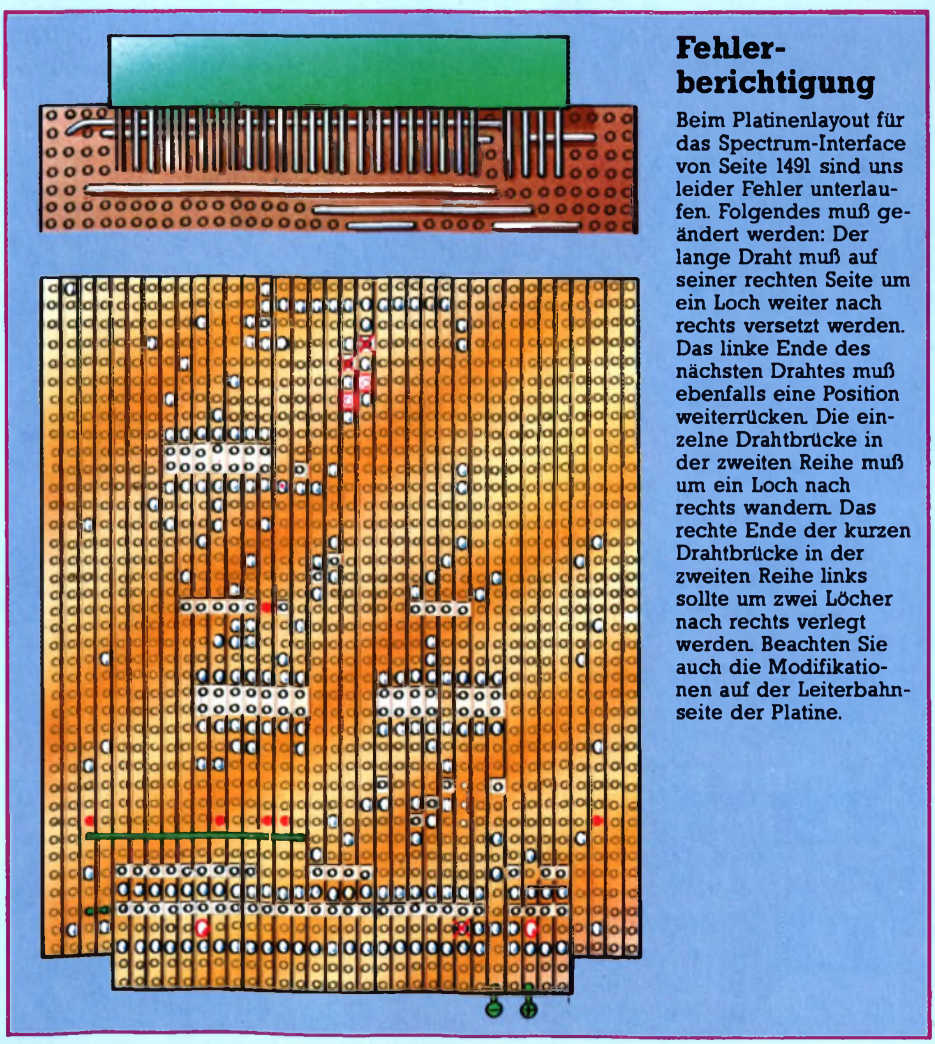
Low-Level Language = Maschinenorientierte Sprache
Eine Programmiersprache stellt zumeist einen Kompromiß zwischen zwei gegensätzlichen Forderungen dar: Einerseits soll sie aus Gründen der Effizienz an der Funktionsweise des Prozessors ausgerichtet, andererseits aber so benutzerfreundlich wie möglich sein. Wenn die Anweisungen eng an den Befehlssatz der CPU angelehnt sind, ist die Bezeichnung „maschinenorientierte“ oder auch „niedere” (Low-Level) Sprache üblich. Dabei ist wegen der Verwandtschaft mit der Kommandostruktur des Prozessors wenig Übersetzungsaufwand erforderlich, wodurch die Arbeit des Rechners entsprechend beschleunigt wird.

Typische maschinenorientierte Sprachen sind neben dem hexadezimalen Maschinencode vor allem die Assemblersprachen. Die Grenze zu den höheren Programmiersprachen ist fließend – beispielsweise lassen FORTH-Anweisungen eigentlich keinen Bezug zum CPU-Befehlssatz erkennen, aber strukturell orientiert sich diese Sprache eng an dem Befehlsablauf des Prozessors.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Bildnachweise
1737–1739: Marcus Wilson-Smith
U2, 1740, 1742, 1743, 1755: Kevin Jones
1744: Caroline Clayton
1747: Mecca Bookmakers
1750, 1751, 1756, 1757, 1760, 1761: Ian McKinnell
1758: Spectrum 1764, U3: Liz Dixon

Machine Independent = Maschinenunabhängig
Software wird meist für ein ganz bestimmtes Rechnermodell mit charakteristischen Eigenschaften geschrieben. Soll ein Programm etwa die Grafikfähigkeiten des Apple voll ausschöpfen, dann muß es wegen der Inkompatibilität der Mikrocomputer für den Commodore 64 vollständig neu gestaltet werden. Die Programmierung kann aber (unter Verzicht auf die Nutzung der Eigenheiten eines speziellen Rechners) auch so erfolgen, daß die Software maschinenunabhängig auf einer Vielzahl von Computern lauffähig ist. Solche Programme werden als „portabel“ bezeichnet; die Portabilität ist im allgemeinen nur zwischen Rechnern mit gleichem Betriebssystem (etwa CP/M) gegeben.



Fehlerberichtigung

Beim Platinenlayout für das Spectrum-Interface von Seite 1491 sind uns leider Fehler unterlaufen. Folgendes muß geändert werden: Der lange Draht muß auf seiner rechten Seite um ein Loch weiter nach rechts versetzt werden. Das linke Ende des nächsten Drahtes muß ebenfalls eine Position weiterrücken. Die einzelne Drahtbrücke in der zweiten Reihe muß um ein Loch nach rechts wandern. Das rechte Ende der kurzen Drahtbrücke in der zweiten Reihe links sollte um zwei Löcher nach rechts verlegt werden. Beachten Sie auch die Modifikationen auf der Leiterbahnseite der Platine.

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Macro = Makrobefehl
Ein Makrobefehl oder kurz Makro ist ein einzelner Befehl innerhalb eines Programms, der für eine ganze Anweisungsfolge steht. Sie wird an anderer Stelle definiert und ausgeführt, wenn das Programm auf den „Namen" des Makros trifft. Makros werden vor allem bei der Assemblerprogrammierung verwendet.

Ein „Makro-Assembler" erlaubt die Definition eines Makro, für die dann im Assemblerprogramm nur ein einziger Befehl erforderlich ist. Insofern sind Makros mit den Prozeduren beim strukturierten BASIC oder bei LOGO verwandt. Die Prozedur wird jedoch als Unterprogramm ausgeführt, während ein Makro bei der Assemblierung jedesmal durch die Folge der zuvor definierten Anweisungen ersetzt wird. Wie ein Prozedurname kann auch ein Makroname, wenn er einmal vereinbart ist, im Programm beliebig oft Verwendung finden.

Magnetic Card = Magnetkarte
Magnetkarten wurden Ende der sechziger Jahre als Speichermedium für die Daten- und Textverarbeitung verwendet, insbesondere bei IBM in Verbindung mit den „Selectric"-Schreibmaschinen. Die Karten aus Kunststoff oder starkem Papier hatten etwa das Format einer Lochkarte und waren ähnlich wie Disketten und Bänder mit einem ferromagnetischen Material beschichtet. Die gespeicherte Information konnte durch einen speziellen Kartenleser wieder entziffert und mit der Schreibmaschine ausgedruckt werden.

Heute sind Magnetkarten vorwiegend als Scheck- und Kreditkarten in Gebrauch. Ein Magnetbandstreifen auf der Kartenrückseite enthält in codierter Form den Namen des Inhabers und seine Bankverbindung mit Kontonummer. Anhand dieser Daten können Bankgeschäfte über Automaten abgewickelt werden. Kleine Kartenleser gibt es auch für Einzelhändler, um Kreditkarten rechnergestützt mit der Liste gestohlener oder verlorgengegangener Karten zu vergleichen und um Zahlungsleistungen sofort über ein Telefonmodem durch einen Zentralcomputer der Bank bestätigen zu lassen.

Verbreitet ist die Verwendung von Magnetkarten bei Bargeldautomaten. Der Magnetstreifen auf der Karte enthält eine Codenummer, die elektronisch gelesen und mit den Ziffern verglichen wird, die der Benutzer eintastet. Stimmen die Zahlen überein, wird der Zugang zur Geldausgabe freigegeben.

Magnetic Disk = Magnetplatte, Diskette
Magnetplatte und Diskette sind Datenträger in Gestalt einer rotierenden Scheibe (Disk), deren Oberfläche mit einer dünnen Aufzeichnungsschicht aus magnetisierbarem Material überzogen ist. Bei der flexiblen Diskette oder ‚Flopy Disk' ist die Platte selbst aus elastischem Kunststoff, bei der starren Magnetplatte oder ‚Hard Disk' im allgemeinen aus einer Aluminiumlegierung. Für die Beschichtung wird bei Disketten meist Eisenoxid verwendet, bei Starrplatten oft auch ein Kobalt-Nickel- oder ein Kobalt-Chrom-Film. Der Scheibendurchmesser variiert von 7,6 cm bei der 3-Zoll-Diskette bis zu 91,4 cm bei Festplatten für Großrechner.

Bildnachweise
1765: Darrell Huff
1778, 1769: Caroline Clayton
1774, 1775, 1786: Ian McKinnell
1777–1779: Crispin Thomas
1783: Mike Clowes
1784: MS-Fortran, Computer World
1785: Alan Adler
1787: BBC Open University Production Centre
1788, 1789, 1791: Kevin Jones
1792: Liz Dixon

Fachwörter von A bis Z

Mail Box = Briefkasten
Ein (elektronischer) Briefkasten ist ein Speicherbereich im Computer zur Aufnahme von Daten, die als „elektronische Post" von einem Computer zum anderen übermittelt werden. Die Sendestation versieht die Nachricht mit der „Adresse" des Empfängers, unter der die Information dann bis zum Abruf verbleibt. Der Inhalt der Mail Box ist gewöhnlich gesichert; vor der Freigabe einer Nachricht muß sich der Empfänger durch ein Paßwort ausweisen. Findet er nach dem Einloggen ins System eine für ihn bestimmte Nachricht vor, kann er sie zum eigenen Computer übertragen und ausdrukken oder auf Diskette abspeichern.

Großrechner sehen heute in der Realität selten wie die raumfüllenden Ungetüme aus, die aus Science-fiction-Filmen bekannt sind. Der berühmte Cray-1, ein Supercomputer der neueren Generation, ist in einer Anzahl von handlichen kleinen Schränken untergebracht, die der arglose Betrachter zunächst eher für Möbelstücke halten würde.

Mainframe = Großrechner
Vor der Microprozessor-Revolution waren Computer äußerst voluminöse und unhandliche Anlagen; die CPU und der Arbeitsspeicher wurden damals als Mainframe bezeichnet. In-

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

zwischen gibt es Rechner der verschiedensten Größen- und Leistungsklassen, und die Bedeutung des Wortes hat sich verschoben.

Die Gruppe der Microcomputer reicht von den Taschenmodellen über die Handheld- und Portable-Typen bis zu den Schreibtisch-PCs. Dann kommen die „Minicomputer" – leistungsfähige Anlagen für Prozeßsteuerung oder Mehrbenutzerbetrieb, die heute meist mit 32-Bit-Prozessoren und einer Hauptspeicherkapazität von 20–40 Megabyte arbeiten. Systeme, die noch eine Stufe schneller sind und über mehr Speicherraum verfügen, werden gelegentlich als „Super-Minis" bezeichnet. Darauf folgen die Mainframes im neueren Sinn: Großrechneranlagen, die in kommerziellen Rechenzentren verwendet werden und etliche Millionen Mark kosten.

Der Begriff Mainframe wird dabei für die gesamte Anlage und nicht nur für CPU und Zentralspeicher verwendet. Umsatzgrößter Hersteller ist IBM, der 70% des Weltmarkts abdeckt und an dem sich die Konkurrenz orientiert. Die Mainframes sind jedoch heute nicht mehr die leistungsfähigsten Systeme.

Management Information System = Management-Informationssystem
Viele große Firmen leisten sich ein „Management-Informationssystem" (MIS), um die nötigen Informationen für Management-Entscheidungen zu sammeln und über Dialog-Auskunftverfahren zugriffsbereit zu halten. Diese Einrichtungen kamen mit dem Einzug der Datenverarbeitung in die Unternehmensführung Ende der sechziger Jahre in Gang. Beim Aufbau eines MIS muß zunächst ein Systemanalytiker die Firmenorganisation unter die Lupe nehmen, um festzustellen, welche Daten jeder Manager benötigt und wie er sie am besten auf Knopfdruck abrufen kann. Wenn das MIS einmal läuft, wird es von einer speziellen MIS-Abteilung betreut, die die Informationen aufbereitet und verwaltet.

Eine Variation des MIS ist das „Decision Support System" (Entscheidungshilfesystem), das mehr Flexibilität bietet. Dabei muß sich jeder Manager seine individuelle Datenbank aufbauen, indem er nach Bedarf Informationen aus dem Zentralrechner der Unternehmen übernimmt. Die Decision Support Systems sind erst durch den vermehrten Einsatz von Arbeitsplatzrechnern möglich geworden, und eine wachsende Anzahl von Entscheidungsträgern macht inzwischen davon Gebrauch, um sich unmittelbar mit „Rohdaten" zu versorgen.

Mass Storage = Massenspeicher
Massenspeicher sind periphere Speicher zur Aufnahme umfangreicher Dateien, deren Platzbedarf die RAM-Kapazität des Rechnersystems übersteigt. Während früher eine Massenspeichergröße von einem Megabyte schon als durchaus ansehnlich galt, haben heute Micros wie der IBM PC oder der Apricot bereits Festplatten mit über zehn Megabyte zu bieten. Zu den Massenspeichern im eigentlichen Sinn zählt beispielsweise das automatisierte Magnetbandsystem 3850 von IBM, das bis zu 472 Gigabyte aufnehmen kann.

Bildnachweise
1793: Liz Heaney
1795, U4: Marcus Wilson-Smith
1800: Mike Cowles
1800–1802: David Vymer
1801, 1810, 1811: Caroline Clayton
1807: Chris Stevens
1808, 1809: Ian KcKinnell
1815–1817: Kevin Jones

Fachwörter von A bis Z

Matrix = Matrix
Eine Matrix ist ein mehrdimensionales System von mathematischen Elementen, die im Computer nach einem bestimmten Schema als Array abgespeichert werden. Jeder Dimension der Matrix wird ein eigener Index zugeordnet, und das einzelne Feldelement ist dann durch einen Satz von Indexwerten eindeutig definiert. Bei einer zweidimensionalen Matrix läßt sich beispielsweise die Variable x als Zeilen- und y als Spaltenindex verwenden; das Wertepaar (2, 3) würde das Element im Kreuzungspunkt der 2. Zeile und 3. Spalte bezeichnen.

Memory Hierarchy = Speicherhierarchie
Jeder Speichertyp benötigt eine andere Zugriffszeit. Deshalb wird die Speicherung der Daten „hierarchisch" organisiert: Die Daten nehmen je nach Zugriffszeit für die Programmierung einen unterschiedlichen Rang ein. Die am schnellsten benötigten Daten stehen in der Hierarchie obenan, und die akut benötigten Daten müssen vom System jeweils rechtzeitig aus den unteren Ebenen hochgeschoben werden.

Die Spitze der Hierarchie bilden bei einem Microcomputer die CPU-Register, die aber insgesamt nur wenige Byte aufnehmen. Danach kommt der Teil des RAM, der zur aktuellen „Zero Page" gehört, gefolgt von den übrigen RAM-Bytes. Darunter rangieren die Daten des gegenwärtig benutzten Datenträgers. Zur untersten Ebene werden schließlich die Disketten und Cassetten gerechnet, die nicht im Abspielgerät stecken.

Die Zugriffszeit wird dabei von oben nach unten immer größer; sie reicht von Mikrosekunden-Bruchteilen für die CPU-Register bis zu Minuten für Cassetten, bei denen erst noch Einlegen und Spulen erforderlich ist. Zum Ausgleich weisen die Speicher auf den untersten Hierarchiestufen die größte Kapazität auf, und zugleich sinken die Kosten der Datenspeicherung. Während Sie bei den CPU-Registern mit etlichen Mark pro Byte rechnen müssen, erhalten Sie für dieses Geld bis zu ein Megabyte an Disketten- oder Cassettenkapazität.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Microprocessor = Microprozessor
Ein Microprozessor ist ein hochintegrierter Baustein, der die CPU des Microcomputers enthält. Der Prozessor umfaßt mindestens das Rechenwerk (ALU) für die Ausführung arithmetisch/logischer Operationen und das Steuerwerk mit Befehlszähler sowie Stack- und Adreßregistern. Im Steuerwerk ist auch der Befehlssatz des Prozessors gespeichert, also die Liste der CPU-Kommandos, und außerdem ist es für die gesamte Ein/Ausgabesteuerung zuständig, also für den Informationsfluß zwischen der CPU, dem Arbeitsspeicher und den anderen angeschlossenen Systemkomponenten.

Der erste bekanntere Microprozessor war der Intel 4004, der aufgrund einer Idee von Edward Hoff Ende der sechziger Jahre entwickelt und ab 1971 gefertigt wurde. Seither sind Hunderte verschiedenster Prozessortypen entworfen und in Millionenauflage produziert worden – die erfolgreichsten waren der 6502 vom MOS Technology, der Zilog Z80 und der Intel 8088. Heute liegt das Hauptgewicht der Entwicklung auf preisgünstigen 32-Bit-Microprozessoren, die sich mit ihren Vorfahren kaum noch vergleichen lassen.

MIDI = MIDI
MIDI ist eine Abkürzung für „Musical Instrument Digital Interface" (Digitale Instrumentenschnittstelle) – ein international anerkannter Standard, der die Kopplung von Computern und elektronischen Musikinstrumenten unterschiedlicher Hersteller ermöglicht. Im MIDI-Verbund kann eine Vielzahl von Instrumenten und Bandgeräten durch einen Steuerrechner synchronisiert werden.

Der zentrale Computer weist jedem Instrument im MIDI-Netz eine Kanaladresse zu. Möchte er irgendwann eine ‚Message' (Mitteilung) an ein bestimmtes Instrument senden, übergibt er sie zunächst dem MIDI-Steuerinterface. Dort wird das Message-Byte mit einem Start- und einem Stopbit versehen, und die zehn Bit werden dann seriell über die MIDI-Leitung geschickt. Wo die Empfängeradresse mit dem Adreßteil der Message übereinstimmt, tritt das MIDI-Interface in Aktion, um den mitgegebenen Befehl zu entschlüsseln und auszuführen; andernfalls wird die Anweisung ignoriert.

Auf dem Markt erscheinen immer mehr MIDI-Schnittstellen, die sich mit den verschiedensten Microcomputern steuern lassen. Zwar verlangt die Mindestspezifikation nur einen MIDI-IN- und einen MIDI-OUT-Anschluß, aber viele Hersteller gehen darüber hinaus. Das hier abgebildete Interface MP401 von Roland bietet eine ganze Reihe zusätzlicher Steuerports.

Bildnachweise
1821: Marcus Wilson-Smith
1823, 1825, 1840, 1841: Ian McKinnell
1827, 1831, 1834, 1835, 1843: Kevin Jones
1836: Crispin Thomas
1837: Epson
1838, 1838: Caroline Clayton
1842: Imagebank

Fachwörter von A bis Z

Minicomputer = Minicomputer
Ursprünglich verstand man unter Minicomputer Rechner von der Größe eines Aktenschranks, im Unterschied zu den Mainframes, die weit mehr Platz beanspruchten. Minicomputer sind langsamer und weniger leistungsfähig als ihre großen Brüder, aber sehr viel preiswerter und daher auch schon in kleineren Betrieben anzutreffen, für deren Erfordernisse Mainframes weit überdimensioniert und zu kostspielig wären.

Einige Fachleute haben angesichts immer leistungsfähigerer Micros das Aussterben der Minicomputer prophezeit. Es scheint aber so, als ob sie sich trotz des drastischen Leistungszuwachses bei den Microprozessoren noch etliche Jahre halten würden.

Modem = Modem
Ein Modem (als Abkürzung für Modulator/Demodulator) ist ein Peripheriegerät für die Rechnerkommunikation über Fernsprechleitungen; es besorgt die Umwandlung der Digitalsignale eines Rechners in übertragbare Analogsignale und umgekehrt. Diese Umsetzung wird in der Fachsprache als Modulation bzw. Demodulation bezeichnet.

Die heute angebotenen Modems lassen sich im wesentlichen zwei Gruppen zuordnen: Bei den Akustikkopplern besteht nur eine akustische Verbindung mit dem Fernsprechnetz durch den auf das Modem gesetzten Hörer, während bei den direkt gekoppelten Modems über einen Zwischenstecker an der Telefonsteckdose ein unmittelbarer elektrischer Kontakt mit dem Postnetz hergestellt wird.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Bei der Demodulation wird aus dem analogen Telefonsignal wieder ein für den Computer verständliches Bitmuster generiert. Außer für die Modulation und Demodulation sind Modems auch dafür eingerichtet, die Baudrate bei der Übertragung anzupassen. Da es bisher keinen verbindlichen Baudratenstandard gibt, sind die Übermittlungsgeschwindigkeiten oft unterschiedlich, und ein Datenaustausch zwischen Rechnern ist ohne Anpassungsvorkehrungen unmöglich.

Mit zunehmender Verbreitung von Microcomputern und Datennetzen werden immer mehr Informationssysteme für Micros mit Modem zugänglich.

Modular Programming = Modulare Programmierung
Modulare und strukturierte Programmierung sind praktisch gleichbedeutend. Gemeint ist damit die Unterteilung eines Programms in Abschnitte, von denen jeder nur für eine Aufgabe zuständig ist, im Gegensatz zum „monolithischen" Programmierstil. Die Modularität erlaubt es, die einzelnen Bausteine unabhängig voneinander zu testen, bis sie einwandfrei laufen. Dann werden sie durch Parameteraustausch untereinander verknüpft, und das Programm arbeitet als Ganzes.

Modulation = Modulation
Modulation wird in der Nachrichtentechnik das Aufprägen von Information auf ein Trägersignal genannt; beim Datenverkehr erfolgt das im allgemeinen durch Umsetzen eines Digitalsignals in ein Analogsignal. Der umgekehrte Vorgang heißt Demodulation. Es gibt verschiedene Modulationsverfahren: Bei der Amplitudenmodulation (AM) wird die Amplitude, das heißt, die Höhe einer als Träger dienenden Wechselspannung verändert, während bei der Frequenzmodulation (FM) die Frequenz des Wechselspannungsträgers variiert wird. Damit verwandt ist die Phasenmodulation (PM), bei der die Information in einer zeitlichen Verschiebung des Trägers enthalten ist. Bei Hochgeschwindigkeits-Modems wird zuweilen ein kombiniertes AM/PM-Verfahren benutzt. Die Pulscodemodulation (PCM) dient dagegen zur digitalen Darstellung eines Analogsignals, dessen jeweilige Höhe in kurzen Zeitabständen abgetastet und durch ein Impulsmuster codiert wird.

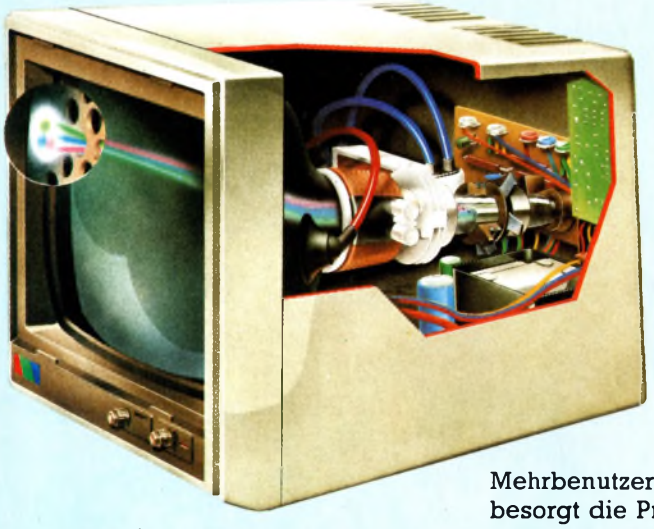
Modulus = Modul
Das Wort ‚Modul' hat mehrere Bedeutungen. Es wird unter anderem in der Mathematik zur Bezeichnung der arithmetischen Operation für das Bestimmen des Teilungsrests – etwa bei der Division A:B – verwendet. Dafür schreibt man „A mod B" (gesprochen „A modulo B"); in PASCAL findet es sich als MOD-Anweisung wieder und wird meist in Verbindung mit dem DIV-Befehl für die ganzzahlige Division (ohne Rest) benutzt. Zur Illustration: 17 DIV 5 ergibt 3, und 17 MOD 5 liefert 2, da bei der Rechnung 17:5=3 der Rest 2 herauskommt.

Bildnachweise
1849: Chris Corr
1850, 1851: Macintosh/Maccharlie
1852, 1853, 1861–1863, U3: Ian McKinnell
1858: Liz Heaney
1864, 1865: Kevin Jones
1868: Liz Dixon
1869, 1873: Caroline Clayton
U4: Rolf Seiffe

Fachwörter von A bis Z

Monitor = Monitor
Als Hardwarebegriff wird die Bezeichnung Monitor für ein Datensichtgerät verwendet, dessen Bildröhre der eines normalen Fernsehapparates entspricht. Die Elektronik ist jedoch speziell für die direkte Wiedergabe des Computer-Bildsignals ausgelegt. Daher bietet der Monitor eine erheblich höhere Bildqualität als ein „zweckentfremdeter" Fernsehempfänger, bei dem die Information ja erst einem hochfrequenten Träger aufmoduliert werden muß, um das übliche Antennensignal zu simulieren.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



Monochrom-Monitore haben meist einen grün oder bernsteinfarben leuchtenden Schirm, der die Augen auch bei längerem Arbeiten am Bildschirm nicht übermäßig belastet. Farbmonitore gibt es in zwei Ausführungen: Bei den RGB-Monitoren werden die Farbanteile Rot, Grün und Blau getrennt übertragen, während die Composite-Video-Monitore mit einem einzigen Mischsignal arbeiten.

Farbmonitore sind entweder für ein „Composite Video" (=FBAS)-Signal oder für RGB-Signale eingerichtet. Beim RGB-Monitor werden die Farbanteile Rot, Grün und Blau über getrennte Leistungsadern und eine mehrpolige DIN- oder auch Euro-AV-Buchse zugeführt. Ein Composite-Video-Monitor empfängt statt dessen nur ein einziges Mischsignal, das im Gerät zur Gewinnung der Farbanteile decodiert werden muß. Der Anschluß erfolgt gewöhnlich über eine BNC-Buchse, wie sie bei Videorecordern gängig ist. Die Bildqualität liegt bei den RGB-Monitoren durchweg eine Stufe höher als bei den Composite-Video-Geräten.

Softwaremäßig versteht man unter „Monitor" ein Systemprogramm, das Steuerungs- und Überwachungsaufgaben wahrnimmt, vorzugsweise beim Abarbeiten von „Jobs" im Mehrbenutzerbetrieb. Der Monitor besorgt die Prioritätssteuerung, die Zuteilung von CPU- und Peripheriezeiten an Simultanprogramme und koordiniert den Datenaustausch. Wo der Monitor für die Organisation des Gesamtsystems zuständig ist, wird er auch als „Supervisor" oder Hauptsteuerprogramm bezeichnet.

Motherboard = Hauptplatine
Das Motherboard ist die Hauptplatine eines Computers. Darauf befinden sich alle wesentlichen Komponenten, etwa die CPU, ein Teil des Arbeitsspeichers und die Ein-/Ausgabesteuerung. Das Motherboard trägt oftmals eine ganze Reihe von Steckleisten für Zusatzkarten.

Als in der Frühzeit der Heimcomputer die Beschäftigung mit Micros noch eine Domäne der Elektronikbastler war, versuchten die Freaks, sich gegenseitig durch Einbau von Zusatzkarten zu übertrumpfen. Mancher Apple platzte buchstäblich aus den Nähten, weil man ihn einfach zu sehr „vollgestopft" hatte. Zwar gibt es das auch heute noch, aber das große Geschäft mit den Zusatzkarten spielt sich im kommerziellen Bereich ab, wo Personal Computer mit anwenderspezifischen Sonderfunktionen ausgestattet werden sollen.

Mouse = Maus
Die Maus ist ein Eingabegerät, mit dem sich durch Hin- und Herfahren auf der Tischplatte der Cursor steuern läßt. Druckknöpfe auf der Maus ermöglichen das Auslösen definierter Rechneraktionen, wenn sich der Cursor über einem bestimmten Piktogramm oder Menüdetail befindet.

Eine solche Maus hat auf der Unterseite eine bewegliche Kugel, deren Rotation sich über Reibrollen auf zwei Codierscheiben überträgt, die ihrerseits durch Lichtschranken abgefragt werden. So wird die Bewegung der Maus in eine elektrische Impulsfolge umgesetzt und an den Rechner übermittelt.

Noch vor zwei Jahren galt die Maussteuerung in Fachkreisen als reine Spielerei. Durch den Erfolg des Apple Macintosh erkannte die Industrie die Vorteile dieses Eingabegeräts und begann, für immer mehr Rechner sogenannte „Maus-Pakete" zu produzieren.

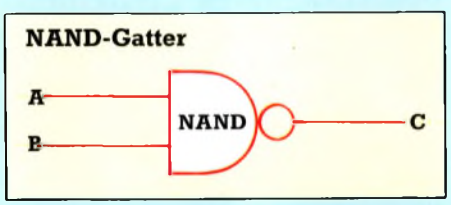
Etwas ganz anderes sind die „intelligenten" Robotermäuse, die sich in einem unbekannten Labyrinth zurechtfinden. Sie verfügen über ein Sensorsystem für das Erkennen von Hindernissen und sind entweder mit einer eigenen Steuerlogik ausgestattet oder sie hängen an einem Rechner, der das „Denken" für sie übernimmt. Die Maus fährt durch das Labyrinth, bis sie irgendwo anstößt, und versucht dann, den Ausgang zu finden.

Bildnachweise
1877: Paul Bryant
1878–1879, 1882: Caroline Clayton
1883: Dimension Graphics
1885: Marcus Wilson-Smith
1888–1895: Rolf Seiffe
1896, 1897: Ian McKinnell
1898, 1899, U3: Kevin Jones

Fachwörter von A bis Z

Name = Name
Namen dienen im Computerbereich zum Identifizieren unterschiedlicher Daten. Namen können in Programmier- und Systemsprachen für Dateien, Prozeduren, Variable, Tabellenfelder und vieles mehr verwendet werden, und zwar nicht nur als Gedächtnishilfe: Der Programmierer braucht sich keine weiteren Gedanken um den Speicherort einer Variablen oder Datei zu machen. Das System legt automatisch Listen mit der Zuordnung von Namen und Adressen an. Wenn der Rechner aus Platzgründen Dateien im RAM oder auf Diskette hin- und herschiebt, ist der Zugriff über die Dateinamen trotzdem stets gesichert, ohne daß dem Programmierer die Verlagerung überhaupt bewußt wird.

NAND = NAND
NAND ist die Abkürzung von NOT AND und bewirkt als Boolesche Verknüpfung eine UNDung mit Negation des Ergebnisses. Die NAND-Funktion liefert daher umgekehrt wie das AND das Resultat FALSE, wenn alle Eingangsgrößen gleichzeitig den Wert TRUE haben. Elektronisch wird die Verknüpfung durch ein NAND-Gatter realisiert, dessen Ausgang genau dann auf 1 liegt, wenn alle Eingänge den Wert 0 führen (in positiver Logik).



Das Schaltzeichen für den NAND-Gatter entsteht aus dem AND-Symbol, indem ausgangsseitig zur Kennzeichnung der Negation ein kleiner Kreis angefügt wird.

Negative Logic = Negative Logik
Wie zu erwarten, stehen bei negativer Logik alle Konventionen auf dem Kopf: Die binäre 1 bedeutet FALSE und die 0 bedeutet TRUE. Die Einführung hat rein technische Gründe – viele Schaltungen lassen sich dadurch einfacher gestalten. Negative Logik ist z. B. bei Microprozessoren für das Schreib/Lese-Signal verbreitet, bei dem die 0 Freigabe und die

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

1 Sperren bewirkt. Der Begriff wird auch verwendet, wenn bei Booleschen Operationen durch Komplementbildung die ursprüngliche Funktion ins genaue Gegenteil verkehrt wird.

Nesting = Verschachtelung
Die Verschachtelung ist ein wichtiges Prinzip bei der Programmierung von Mehrfach-Schleifen. In BASIC muß jede innere FOR-Schleife vollständig in der äußeren Schleife eingebettet sein. Eine Verletzung dieser Grundregel löst bestenfalls eine Fehlermeldung aus, meist aber einen Programmabsturz. Wenn der Rechner auf das Schlüsselwort FOR trifft, schiebt er die zugehörige Adresse auf den Stack. Stößt er dann auf die folgende NEXT-Anweisung, holt er sich die FOR-Adresse wieder und verzweigt zum Schleifenanfang. Wenn Schleifen nicht korrekt geschachtelt sind, lädt der Computer die falsche Rücksprungadresse vom Stack und setzt die Programmbearbeitung an einer falschen Stelle fort.

Die Schachtelung wird vor allem akut, wenn wiederholt gleichartige Operationen auszuführen sind, wie etwa in dem folgenden Programm:

```
10 FOR A=1 TO 12
20 FOR B=1 TO 12
30 PRINT A;"x";B;"="; A*B
40 NEXT B
50 NEXT A
```

Die Ausführung beginnt mit der A-Schleife, indem die Adresse der ersten FOR-Anweisung auf den Stack gelegt und irgendwo im Speicher die Variable A auf 1 gesetzt wird. Das gleiche Arbeitsprinzip gilt für die B-Schleife. Da der Stack nach dem FIFO-Prinzip organisiert ist, lädt der Rechner dann in Zeile 40 gemäß der NEXT-Anweisung als oberste Stack-Adresse wieder die des FOR von Zeile 20 und setzt das Programm dort mit der Erhöhung der Variablen B fort. Nach zwölfmaligem Durchlaufen der B-Schleife und anschließendem Rücksprung auf Zeile 20 erreicht B den Wert 13, das heißt, der Schleifenendwert ist überschritten. Daraufhin geht der Rechner auf Zeile 50 über, wo er wieder ein NEXT findet, sich daraufhin vom Stack die Einsprungsadresse der A-Schleife greift usw.

Verschachtelte Schleifen

```
5 REM Korrekte Schachtelung
10 FOR X=0 TO 6
20 FOR Y=0 TO X
30 PRINT X;":";"Y";"";
40 NEXT Y
50 PRINT
60 NEXT X

5 REM Fehlerhafte Schachtelung
10 FOR X=0 TO 6
20 FOR Y=0 TO X
30 PRINT X;":";Y;"";
40 NEXT X
50 PRINT
60 NEXT Y
```

Bei korrekter Schachtelung muß die innere Schleife ganz in die äußere eingebettet sein. Beim oberen Programm liegt die Y-Schleife vollständig innerhalb der X-Schleife, und es gibt keine Probleme. Im unteren Programm ragt dagegen die eine Schleife einseitig über die andere hinaus, und daher sind bei der Ausführung Fehler zu erwarten.

Viele BASIC-Interpreter verlangen den Namen der Laufvariablen in der NEXT-Anweisung nicht. Sie springen ohne Überprüfung auf die FOR-Anweisung zurück, die gerade im Stack ansteht. Eine Fehlschachtelung wäre geradezu fatal.

Bildnachweise
1905, 1907–1910, 1918,
1923–1925: Kevin Jones
1906, 1914: Caroline Clayton
1917, 1919: Chris Stevens
1920, 1921, 1928: Ian McKinnell
U3: Liz Dixon

Fachwörter von A bis Z

Network = Netzwerk
Als Netzwerk wird in der Datenverarbeitung ein Kommunikationssystem zur Verknüpfung mehrerer Rechner und Terminals bezeichnet. Der Datenverkehr kann dabei über Telefonleitungen, Richtfunkstrecken oder Satelliten erfolgen – die räumliche Ausdehnung spielt keine Rolle. Ein Netzwerk kann sich insgesamt auf nur einige PCs im gleichen Büro beschränken oder eine Anzahl Großrechner umfassen, die über die ganze Welt verstreut sind.

Die Bedeutung von Kommunikationsnetzen nimmt auch im Microcomputerbereich ständig zu. Benutzer können im Netzverbund nicht nur untereinander Daten austauschen, sondern bekommen vor allem Zugang zu der enormen Verarbeitungskapazität von Großrechnern und den zugehörigen umfangreichen Informationsangeboten von Datenbanken.

Computernetze werden in zunehmendem Umfang in der Ausbildung von Computeranfängern eingesetzt.

In der Schaltungstechnik versteht man unter einem Netzwerk eine abgegrenzte Bauelement-Gruppe mit spezifischen Eigenschaften. Unterschieden wird dabei zwischen „aktiven" Netzwerken mit verstärkenden Komponenten wie Transistoren und „passiven" Netzwerken, die nur passive Bauteile wie Kondensatoren und Widerstände enthalten. Wenn zwischen Ein- und Ausgangssignal eine lineare Beziehung besteht, spricht man von einem „linearen", andernfalls von einem „nichtlinearen" Netzwerk.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

NOR = NOR
NOR ist eine Abkürzung für die logische Operation NOT OR, also das negierte ODER – eine der sechs Booleschen Verknüpfungen, die in der Schaltalgebra Verwendung finden. Ein NOR-Gatter liefert den Ausgangswert TRUE, wenn beide Eingänge FALSE sind, und verhält sich damit umgekehrt wie ein NAND-Gatter. Werden Eingänge und Ausgang eines NOR-Gatters invertiert, ergibt sich die Funktion eine NAND-Gatters, so daß sich wie mit NANDs auch mit NOR-Gattern sämtliche Booleschen Verknüpfungen realisieren lassen.

NOT = Nicht
Das NOT oder die Inversion ist eine weitere Boolesche Operation. Sie bewirkt die Umkehrung eines logischen Zustands in sein Gegenteil – aus TRUE wird FALSE, aus FALSE wird TRUE. Im Unterschied zu den andern Logikgattern gibt es bei einem NICHT-Gatter nur einen einzigen Eingang.

Numerical Analysis = Numerische Auswertung
Bei der numerischen Bearbeitung mathematischer Aufgabenstellungen wird die Lösung durch Zahlenrechnen gesucht. Dabei sind meist Unmengen von Zahlen zu verarbeiten. Zum Bereich der numerischen Mathematik gehört auch die Optimierung der Vorgehensweisen mit dem Ziel, möglichst effiziente und allgemein anwendbare Verfahren zu entwickeln.

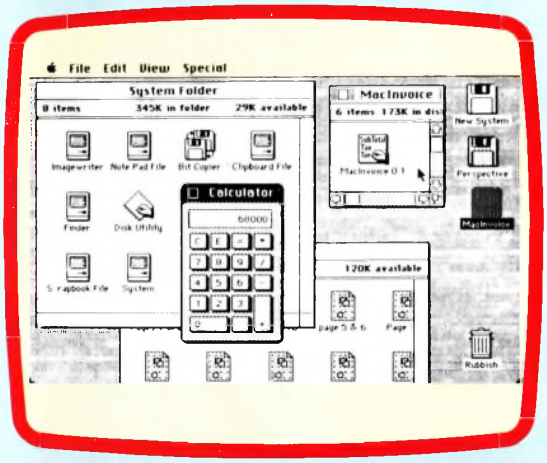
Network Architecture = Netzarchitektur
Die „Architektur" umfaßt die formale und funktionale Struktur eines Kommunikationsnetzes. Funktionale Gesichtspunkte sind unter anderem die Wahl des Übertragungsverfahrens einschließlich Codes und Fehlerkontrolle, die Datenflußsteuerung und die Art der Adressierung der einzelnen Netzknoten. Für den formalen Aufbau ist auch der Begriff „Netzwerktopologie" gebräuchlich – typische Konfiguration sind etwa das Ring- und das Sternnetz.

Noise = Rauschen
Bei den Nachrichtentechnikern und Elektrotechnikern heißt jeder unerwünschte Signalbeitrag Rauschen. Schon beim Schaltungsentwurf ist darauf zu achten, daß Rauscheinflüsse, die sich dem eigentlichen Signal störend überlagern, so klein wie möglich bleiben. Bei Digitalbausteinen der TTL-Familie z. B. wird die logische Eins durch eine Spannung von +5 V, die logische Null durch 0 V dargestellt. Rauschbeiträge können das Signal so stark verfälschen, daß die Zustände „1" und „0" an irgendeinem Gattereingang nicht mehr eindeutig erkennbar sind und Fehler auftreten. Es ist ein wesentliches Ziel der Elektronikentwickler, möglichst hohe „Störschwellen" zu erreichen, das heißt hohe Grenzwerte für die Rauschspannung, die noch ohne Beeinträchtigung der Zuverlässigkeit toleriert wird. Wenn Schaltkreise sehr viel Rauschen ohne Fehlfunktion verkraften, spricht man auch von extremer „Störsicherheit".

Bildnachweise
1933: Paul Chave
1934, 1953, 1958, 1959: Kevin Jones
1938: Liz Dixon
1940: Chris Stevens
1942, 1952: Mike Clowes
1943: Marcus Wilson-Smith
1946, 1949: Graeme Harris
1955: Liz Heany
1956: Caroline Clayton
1960: Tony Sleep

Fachwörter von A bis Z

Object Code = Objektprogramm
Aus dem Quellprogramm, das in einer höheren Programmiersprache geschrieben wurde, erzeugt ein Compiler ein Objektprogramm (in maschinennaher Sprache). Es steht nach dem Compilieren als Maschinenprogramm zur Verfügung und kann mit einem RUN-Befehl gestartet werden. Der Vorteil beim Erstellen eines solchen Objektprogramms liegt darin, daß es wesentlich schneller läuft als etwa ein BASIC-Programm, das während der Ausführung Zeile für Zeile vom Interpreter in die Maschinensprache übersetzt wird. Noch effizienter als vom Compiler erzeugte Objektprogramme arbeiten allerdings Programme, die unmittelbar im Maschinencode verfaßt sind.



Der Apple Macintosh war einer der ersten Rechner, bei dem die objektorientierte Programmierung in die Praxis umgesetzt wurde. Statt eine Serie von Befehlen einzugeben, braucht der Benutzer hier nur den Cursor mit der Maus auf das betreffende Piktogramm zu führen und die Kopftaste der Maus zu drücken. Der Rechner erledigt die gestellte Aufgabe dann ohne weiteres Zutun allein.

Object-Oriented = Objektorientiert
Normalerweise werden beim Programmieren alle Verarbeitungsschritte in Gestalt arithmetischer Anweisungen, Funktionen und Prozeduren vorgeschrieben. Bei der objektorientierten Programmierung wird der Ablauf dagegen nicht im Detail fixiert, sondern es wird in Form einer „Message" (Mitteilung) nur das zu lösende Problem spezifiziert. Der Rechner wählt den Lösungsweg.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Octal Notation = Oktalschreibweise
Im Oktalsystem dient die Zahl Acht als Basis. Es gibt darin also nur die Ziffern Null bis Sieben. Jede Oktalstelle faßt deshalb gerade eine dreistellige Binärzahl. Das Oktalsystem war sehr praktisch für die alten „Stellenmaschinen", die nicht mit Bytes zu acht Bit, sondern mit „Stellen" zu sechs Bit als kleinster adressierbarer Einheit arbeiteten, weil jede „Stelle" als zweiziffrige Oktalzahl (Höchstwert 77≙ dez. 63) darzustellen war. Mit dem Einzug der Bytemaschinen wurde das Oktalsystem durch das Hexadezimalsystem mit der Basis 16 verdrängt.

One's Complement = Einer-Komplement
Das Einer-Komplement ist eine Zwischenstufe bei der Bildung eines Zweier-Komplements, das für die Binär-Arithmetik benötigt wird. Das binäre Einer-Komplement ergibt sich einfach durch Vertauschen von Nullen und Einsen, also durch „Invertieren" (oder NOT-Verknüpfung) einer Zahl; aus 0001 0011 (dezimal 19) wird damit 1110 1100. Wenn der Computer dieses Bitmuster wie üblich als ganze Zahl mit Vorzeichen und daher die „1" an erster Stelle als „Minus" liest, interpretiert er es als -19. Durch Erhöhen um Eins entsteht aus dem Einer- das Zweier-Komplement, dessen Addition bei Digitalrechnern die Subtraktion ersetzt.

Operand = Operand
Als Operanden werden speziell bei der Maschinencode- und Assemblerprogrammierung Größen bezeichnet, auf die „Operationen" wie Addition oder Subtraktion angewandt werden. In der CPU müssen die Operanden zur Ausführung arithmetisch/logischer Befehle in den vorgesehenen Registern bereitstehen.

Operating System = Betriebssystem
Das Betriebssystem ist eine Art Software-Schnittstelle zwischen Benutzer und Rechnerelektronik. Es gibt eine Vielzahl unterschiedlicher Systeme, die sich aber formal leicht zwei Kategorien zuordnen lassen. Die eine umfaßt die ROM-gespeicherten Systeme, die vor allem bei Heimcomputern häufig eingebaut sind.

Die zweite Gruppe bilden Betriebssysteme, die nach dem Einschalten des Rechners erst von einem externen Speicher geladen werden müssen, wie etwa CP/M und MS-DOS. Dabei handelt es sich meistens um diskettenorientierte Systeme. Ihr Vorteil besteht insbesondere darin, daß sie ein hohes Maß an Kompatibilität zwischen hardwaremäßig vergleichbaren Computern gestatten. Bei Maschinen, die mit dem gleichen Betriebssystem arbeiten, sind im Prinzip auch die dafür geschriebenen Softwarepakete von einem zum andern Rechner übertragbar oder „portabel".

Operation = Operation
Durch eine mathematische Operation wird aus einer oder mehreren Ausgangsgrößen ein neues Ergebnis gewonnen. Die Symbole +, − und * beispielsweise repräsentieren arithmetische Operationen. Daneben spielen in der Datenverarbeitung aber noch weitere eine Rolle, etwa binäre Operationen, die letztlich alle auf der simplen Addition zweier Binärzahlen beruhen. Häufig benutzt werden auch Boolesche Operationen, bei denen Operanden durch logische Funktionen verknüpft werden.

Bildnachweise
1961, 1966: Kevin Jones
1967, 1972: Mike Clowes
1970: Marcus Wilson-Smith
1974, 1982: Chris Stevens
1988: Caroline Clayton

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Optical Disc =
Optische Speicherplatte
Für dieses Speichermedium ist der Name ‚Laserplatte' geläufiger. Die digitale Information ist dabei in Form winziger Vertiefungen (Pits) auf konzentrischen oder spiraligen Spuren mit einem starken Laser in die spiegelnde Plattenoberfläche „eingebrannt". Bei der fotoelektrischen Abtastung mit einem schwachen Laser werden die Pits, die das Licht streuen, als Einsen interpretiert, die Reflexe der unversehrten Oberflächenbereiche dagegen als Nullen.

Laserplatten werden derzeit hauptsächlich als ‚Compact Disk' (CD) alternativ zur Schallplatte bzw. als Bildplatte für die Video-Wiedergabe eingesetzt. In Gestalt der ‚CD-ROMs' eröffnet diese Technologie aber auch im Computerbereich weitreichende Möglichkeiten. Eine einzige Compact Disk kann bis zu zwei Gigabyte speichern.

Für Laserplatten bieten sich im Computerbereich interessante Einsatzmöglichkeiten – nicht nur für die große Videoplatte zur Bildwiedergabe in Verbindung mit Computergrafik, sondern auch für die Compact Disk als ‚CD-ROM' zur Speicherung von Programmen. Der jetzige Stand der Technik gestattet allerdings noch nicht das Löschen und Beschreiben der Platten durch den Benutzer.

Die Sache hat einstweilen noch den Haken, daß sich die Platten vom Anwender weder löschen noch beschreiben lassen; sie sind daher vorerst nur für käufliche Software und nicht als Massenspeicher für den Schreib/Lese-Betrieb geeignet.

Bei Spielhallenautomaten werden Bildplatten schon seit längerer Zeit eingesetzt. Sie können dort beispielsweise Ihr Flugzeug (als Rechner-erzeugtes Sprite) durch eine ‚echte' Filmlandschaft von der Laserplatte steuern, die den Sprites als Hintergrund unterlegt wird. Das ergibt eine Wirklichkeitsnähe, die sonst nicht zu erreichen ist.

Optimisation = Optimierung
Die ‚Optimierung' eines Programms zielt auf die Lösung, die den gestellten Anforderungen am besten gerecht wird. Je nach Lage der Dinge kann die optimale Lösung sehr unterschiedlich aussehen – einmal soll die Geschwindigkeit und ein anderes Mal vielleicht die Speicherplatznutzung Vorrang haben.

Zur Optimierung gehört die Ausschaltung überflüssiger Umwege bei der Programmausführung. Ein gutes Beispiel dafür ist die Initialisierung von Variablen außerhalb von Schleifen oder häufig benutzten Unterprogrammen: Wenn einer Variablen einmal ein Wert zugewiesen ist, kostet es nur Zeit, die Vereinbarung immer wieder unnötig zu bestätigen, solange keine Veränderung eintritt.

OR = ODER
Das OR als Boolesche Verknüpfung liefert dann den Ausgangswert ‚TRUE', wenn ein oder beide Eingänge auf ‚TRUE' liegen. Gebräuchlich ist dafür auch die Bezeichnung ‚inklusives' OR zur Unterscheidung vom ‚exklusiven' OR (EXOR), bei dem nur einer von den beiden Eingängen den Wert ‚TRUE' führen darf, wenn sich der Ausgangszustand ‚TRUE' ergeben soll.

Ordered Pair =
Geordnetes Zahlenpaar
‚Geordnet' heißt ein Zahlenpaar dann, wenn die Reihenfolge der Elemente fixiert ist. Bei der Angabe kartesischer Koordinaten in der Form (x,y) beispielsweise, wird stets die erste Zahl als horizontaler, die zweite als vertikaler Achsenabschnitt interpretiert, so daß die Zuordnung nicht unbedingt jedesmal neu spezifiziert werden muß.

Oscilloscope = Oszilloskop
Ein Oszilloskop (auch Oszillograf) dient zur Darstellung des zeitlichen Verlaufs von elektrischen Signalen, speziell von schnellen Schwingungsvorgängen. Die Kurvenform wird dabei mit einem Elektronenstrahl auf dem Leuchtschirm einer Kathodenstrahlröhre geschrieben. Der Strahl wird jedoch anders als beim zeilenweisen Aufbau eines Fernsehbildes durch die Signalspannung nicht helligkeitsmoduliert, sondern vertikal abgelenkt, während er mit einer wählbaren Geschwindigkeit von links nach rechts über den Schirm läuft. So ergibt sich eine Leuchtspur, die den Spannungsverlauf als Funktion der Zeit wiedergibt. Oszilloskope sind im Labor und beim Service ein unerläßliches Hilfsmittel, ob es nun um die Analyse eines Schallsignals oder die Synchronisation von Schaltimpulsen geht.

Bildnachweise
1992: Marcus Wilson-Smith
1993, 2016: Kevin Jones
1997, U3: Ian McKinnell
2000–2005: Garry Banks
2010: Caroline Clayton

Fachwörter von A bis Z

Output Device = Ausgabegerät
Jede Peripheriekomponente, die vom Rechner verarbeitete Informationen für den Benutzer zugänglich macht, wird als Ausgabegerät bezeichnet. Für Microcomputer sind alle möglichen Ausgabegeräte verfügbar; am gängigsten sind Bildschirmgeräte in Gestalt von Monitoren, Fernsehern oder Drucker. Der technische Fortschritt und das zunehmende Eindringen von Rechnern in immer neue Anwendungsfelder bringen es mit sich, daß das Angebot an Ausgabegeräten ständig vielfältiger wird. Die industrielle Fertigung z. B. bedient sich in wachsendem Ausmaß rechnergesteuerter Roboter, und für die nächste Zukunft sagen die Fachleute bei der Ausgabe-Peripherie vor allem einen Entwicklungsschub auf dem Gebiet der Sprachsynthese voraus.

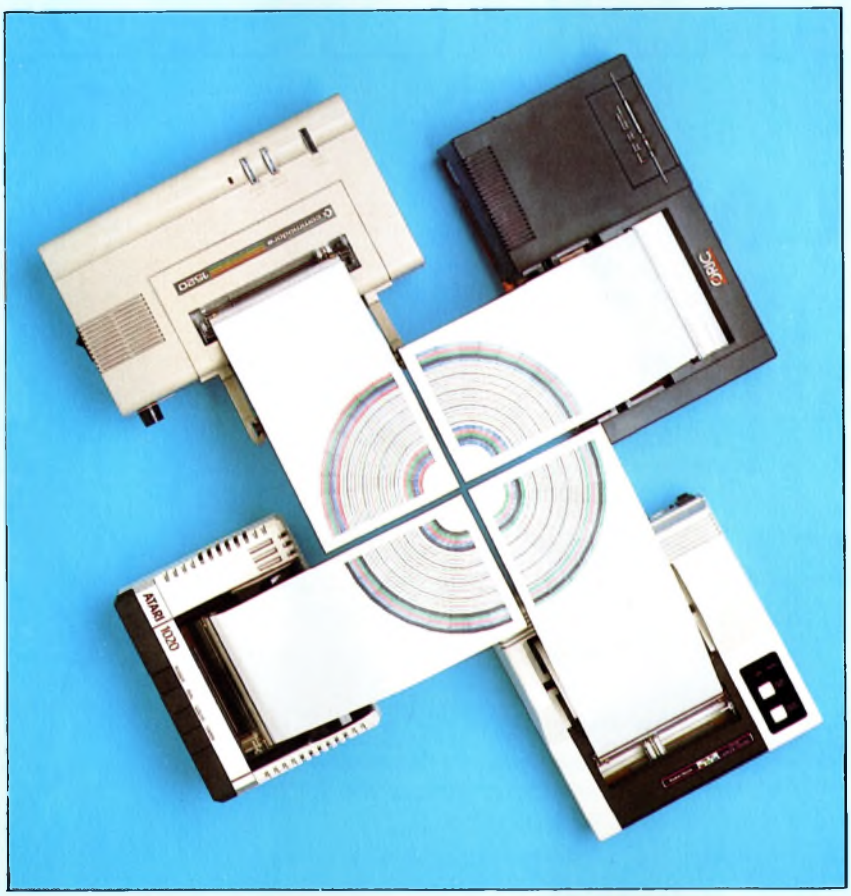
Overflow = Überlauf
Ein Überlauf tritt ein, wenn das Ergebnis einer Rechenoperation die vorgesehene Bitkapazität überschreitet; der Begriff wird dabei auch für den „übergelaufenen" Informationsteil selbst verwendet. Der Computer reklamiert den Überlauf entweder als Fehler, oder aber er beachtet ihn nicht – da beides zu Fehlergebnissen führt, sollte das Auftreten eines Überlauf von vornherein vermieden werden.

Packing = Packen
Beim Packen werden Daten aus Speicherplatzgründen so eng wie möglich zusammengeschoben. Dafür kann sich natürlich jeder Programmierer seine eigenen Tricks einfallen lassen, aber es gibt ein paar bewährte Verfahren: Beim Packen von Dezimalzahlen werden beispielsweise je zwei Ziffern in einem Byte untergebracht; Boolesche Variable, die ja nur ein Bit benötigen, passen zu acht in ein Byte, und beim „Tokenizing" werden häufig auftretende Informationseinheiten durch einen Kurzcode verschlüsselt. Generell kostet das Packen zusätzliche Entflechtungszeit bei der jeweiligen Programmausführung.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Bildnachweise
2017, 2018: Science Photo Library
2019, 2020, 2035: Caroline Clayton
2022: Liz Heany
2028, 2040, 2041, 2043: Kevin Jones
2029, 2031, 2036, U3: Chris Stevens
2039: BBC Hulton Picture Library

Page Printer = Seitendrucker
Ein „Seitendrucker" gibt ähnlich wie eine Druckerpresse in einem Arbeitsgang gleich eine ganze Seite an Information aus, während ein Zeilendrucker (Line Printer) ganze Zeilen, ein serieller Drucker (z. B. mit Typenrad) dagegen nur einzelne Zeichen nacheinander zu Papier bringt. Seitendrucker arbeiten ohne mechanischen Anschlag und können mehrere DIN-A4-Seiten pro Sekunde liefern. Sie funktionieren fast durchweg elektrostatisch, wobei durch einen Laser auf einer Halbleitertrommel ein Ladungsbild erzeugt wird. Dies läßt sich durch Tonerpulver auf Papier übertragen und dort einbrennen. Die sog. „Laserdrucker" gibt es mittlerweile schon zum durchaus für viele User interessanten Preis eines Personal-Computers.



Besonders nützlich sind Ausgabegeräte wie diese „Printer-Plotter", die nicht nur Schriftzeichen, sondern auch Grafik wiedergeben können, und zwar mit Hilfe von Kugelschreiberpatronen. Die Daten vom Rechner werden zweidimensional in Papiervorschub und Schreibkopfbewegung umgesetzt, womit sich beliebige Muster erzeugen lassen.

Fachwörter von A bis Z

Password = Paßwort
Im kommerziellen Bereich wird bei Computersystemen zur Benutzeridentifikation ausgiebig von ‚Paßwörtern' Gebrauch gemacht. Das sind Buchstaben- und Zeichenkombinationen, die beim ‚Einloggen' eingegeben werden müssen, um zu verhindern, daß Unbefugte kostbare Rechenzeit stehlen oder Zugang zu vertraulichen Dateien haben.

Wer sich bei einem derartig geschützten System einloggen (anmelden) will, wird vom Rechner zunächst nach dem Paßwort gefragt. Es ist im Speicher hinterlegt, und der Benutzer erhält nur Zutritt, wenn seine Eingabe mit dem Muster exakt übereinstimmt. Darüber hinaus können bestimmte Systembereiche noch durch Paßworte gesperrt sein.

Bei vielen Datenbanken lassen sich die Paßwörter auch vom Benutzer selbst definieren. Für das erste Einloggen wird zwar eine Kombination zugeteilt, aber danach ist ein beliebiger Austausch möglich.

Manche User haben offensichtlich Schwierigkeiten, sich geeignete Paßwörter einfallen zu lassen. Vielfach werden schlicht Name und Geburtstag des Benutzers genommen, unter Umständen auch der angesteuerte Datenbank-Untertitel. Der verbreitete Phantasiemangel ist ein Segen für die ‚Hacker' und kommt ihnen beim Knacken von Computersystemen außerordentlich zustatten.

Patch = Flicken
Wenn ein Programm nicht richtig laufen will, ist ein nachträglich angehefteter ‚Patch' (Flicken) aus ein paar Korrekturbefehlen manchmal unentbehrlich. Professionelle Programmierer runzeln bei solchem Verfahren allerdings die Stirn und lassen einen ‚Patch' höchstens zu, wenn nur eine Kleinigkeit in einem fertigen Programm auszubessern ist. Der bessere Weg ist natürlich, gleich sauber strukturiert und zuverlässig zu programmieren. Das erspart meist die Flickarbeit; andernfalls führt die Korrektur der Korrekturen leicht zum totalen Chaos.

Hardwaremäßige ‚Patches' sind die fliegenden Drähte, die zuweilen einer Schaltung auf die Beine helfen müssen, wenn die Platine einen kleinen Fehler enthält. Gebräuchlich ist der Begriff auch für die Kabelbrükken, mit denen sich auf sogenannten ‚Patchboards' (Steckbrettern) kurzfristig Verbindungen herstellen lassen.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Pattern Recognition = Mustererkennung
Die Mustererkennung ist ein wichtiges Forschungsgebiet im Bereich der Künstlichen Intelligenz. Der Begriff umfaßt eigentlich jede Form der automatischen Auswertung von Informationsmustern, wird aber meist für die visuelle und die Spracherkennung verwendet. Da der Computer weder Augen noch Ohren hat, müssen für ihn optische oder akustische Eindrücke zunächst mit Hilfe von Kamera bzw. Mikrofon in elektrische Signale umgesetzt werden. Der Erkennungsprozeß beruht dann auf der Deutung dieser Information anhand eines Vergleichs mit im Rechner gespeicherten Vorgaben.

Der Apricot Portable ist als einer der ersten Rechner von vornherein für die Spracheingabe eingerichtet. Einfache Wörter wie Befehle und Ziffern nimmt der Rechner über das eingebaute Mikrofon zur Kenntnis.

Wenn dabei versucht wird, das unbekannte Objekt mit vorrätigen Schablonen zur Deckung zu bringen, spricht man vom ‚Klassifikationsverfahren'. Dem Rechner wird beispielsweise die Form eines Quadrats einprogrammiert, mit der sich dann mehr oder weniger direkt ein Vergleich durchführen läßt. Obwohl diese Methode am einfachsten erscheint, führt sie bei komplexen Mustern zu erheblichen Problemen.

Beim ‚analytischen' Verfahren wird das Muster nicht als Einheit betrachtet, sondern es wird auf seine Komponenten hin analysiert. Im genannten Beispiel könnte der Rechner das Muster daraufhin untersuchen, ob seine Ecken rechtwinklig und alle Kanten gleich lang sind, so daß es sich als Quadrat einordnen läßt. Dieses Verfahren erfordert zwar wesentlich mehr Programmieraufwand, ist aber universeller.

Wenn der Rechner überhaupt erst einmal bestimmte Muster identifizieren kann, ist als nächstes dafür zu sorgen, daß er sie auch bei gewissen Abweichungen von der starren Schablone noch wiedererkennt. Das spielt besonders bei der Spracherkennung eine entscheidende Rolle. Bei der Sprachanalyse wird das Frequenzspektrum des akustischen Signals untersucht.

Bildnachweise
2045: Crispin Thomas
2046: MTV
2047: British Telecom
2049, 2051, 2054, 2066,
2069: Caroline Clayton
2050: Hil Rudge
2054, 2061, 2067, 2071: Kevin Jones

Fachwörter von A bis Z

Permutation = Permutation
Die Elemente einer Menge können unterschiedlich arrangiert werden; jede mögliche Abfolge bildet dabei eine „Permutation". Für die Buchstaben a, b und c etwa sind sechs Anordnungen denkbar: abc, acb, bac, bca, cab und cba. Allgemein ist die Anzahl der Permutationen von n Elementen durch n! (gesprochen „n Fakultät") gegeben. Die Definition der Fakultät lautet
n! = n * (n—1) * (n—2) * . . . * 1
Mit fünf Elementen lassen sich demnach 5! = 5 * 4 * 3 * 2 * 1 = 120 Permutationen erzeugen.

Picture Processing = Bildverarbeitung
Die Bildverarbeitung beschäftigt sich mit der (meist digitalen) Analyse und Umformung der Signale von bildgebenden Systemen, beispielsweise von Fernsehkameras; die Wiedergabe kann über einen Monitor oder einen geeigneten Drucker erfolgen. Der Informationsgehalt eines Bildes hängt einerseits von der geometrischen Auflösung (Ortsauflösung) ab, das heißt von der Anzahl der Bildpunkte. Ebenso wichtig ist aber zudem die Kontrastauflösung.

Das Bild steht im Speicher als zweidimensionales Array, wobei jedem Pixel ein oder mehrere Bits für die Darstellung von Helligkeit und Farbe zugeordnet sein können. Im einfachsten Fall gibt es nur zwei Helligkeitswerte, nämlich Pixel „ein" oder „aus", je nachdem, ob im Bild eine bestimmte Schwellintensität überschritten ist oder nicht. Mehr Information vermittelt eine Graustufendarstellung, bei der jedem Pixel differenzierte Helligkeitswerte zugewiesen werden können.

Die Bildverarbeitung führt zu einer gezielten Veränderung des Bildcharakters, typisch durch digitale „Filterung" der Bildinformation. Ein Beispiel dafür ist die Verbesserung der Detaildarstellung bei den Raumsonden-Aufnahmen vom Halleyschen Kometen, aber ebenso die Erzeugung verblüffender Effekte bei Videofilmen.

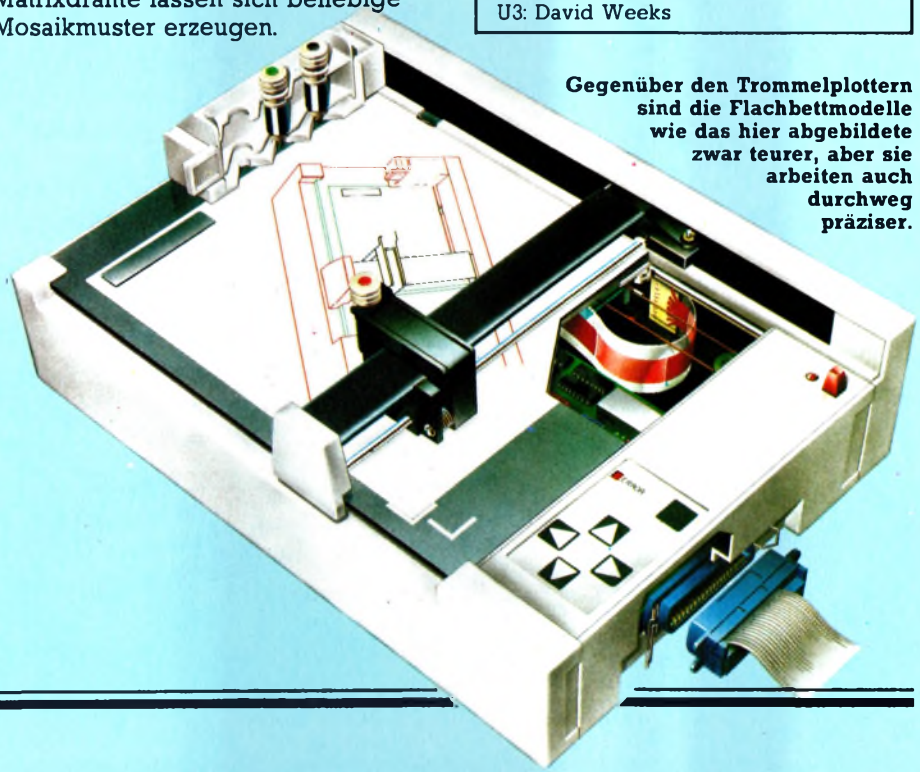
Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Plasma Display = Plasma-Anzeige
Plasma-Anzeigefelder finden bei Computern zunehmend Verwendung. Sie arbeiten mit einer ähnlichen Zeichengeometrie wie Flüssigkristall-Anzeigen und emittieren rotes oder oranges Licht. Es entsteht durch eine elektrische Gasentladung innerhalb eines Elektrodenrasters, das zwischen Glasplatten eingebracht ist. Das Raster wird von dünnen Drähten gebildet, die in Matrixform angeordnet sind. Zwischen ihnen zündet beim Anlegen von Spannung die elektrische Entladung, und die Gasmoleküle werden durch Stoßionisation zum Leuchten angeregt. Durch entsprechende Ansteuerung der Matrixdrähte lassen sich beliebige Mosaikmuster erzeugen.

Plotter = Plotter
Ein Plotter ist ein Zeichengerät, das Rechnerdaten als Grafik zu Papier bringt – im allgemeinen mit Hilfe eines Schreibstifts, der in x- und y-Richtung über das Papier fährt und gehoben oder gesenkt werden kann. Es gibt zwei Bauformen: Flachbettplotter und Trommelplotter. Bei der Flachbett-Ausführung liegt das Papier als Einzelblatt auf einem ebenen Papiertisch, über den sich in x-Richtung ein Zeichenarm bewegt. Auf dem Arm ist in y-Richtung die Schreibstifthalterung verschiebbar angebracht. Außer diesem Kreuzschlittensystem gibt es sogenannte „Zeichenroboter" nach dem Turtle-Prinzip.

Im Heimcomputerbereich sind jedoch die Trommelplotter am stärksten verbreitet. Dort wird ähnlich wie bei einer Schreibmaschine das Papier durch eine Walzenführung in y-Richtung transportiert, während davor in x-Richtung ein Wagen mit dem Schreibstift hin- und herfährt.

Bildnachweise
2073: Tony Sleep
2074, 2075, 2089, 2091: Caroline Clayton
2092: Dimension Graphics
2078, 2079: Liz Heaney
2081, 2093, 2094, 2098, 2100: Kevin Jones
2084: Liz Dixon
2086, 2087: Chris Stevens
U3: David Weeks



Gegenüber den Trommelplottern sind die Flachbettmodelle wie das hier abgebildete zwar teurer, aber sie arbeiten auch durchweg präziser.

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

POKE = POKE
Der Befehl POKE erlaubt es, in einer höheren Programmiersprache, etwa in BASIC, eine vorzeichenlose Zahl auf eine beliebige Adresse (meist ein Register) zu laden. Dazu gibt man nacheinander POKE, die Adresse und schließlich die gewünschte Zahl ein.

POKE ist sehr vielseitig verwendbar; es wird vorwiegend zur Ausführung maschinennaher Operationen benutzt, für die es in der höheren Sprache keine entsprechenden Befehle gibt. Der Commodore 64 verfügt beispielsweise in seinem BASIC nicht über Kommandos, die die weitreichenden Ton- und Grafikmöglichkeiten unterstützen. Die Hardware läßt sich daher nur richtig nutzen, indem bestimmte VIC (Video)- und SID(Klang)-Bausteinadressen mit den entsprechenden Parametern geladen werden.

Polish Notation =
Polnische Notation
Von dem polnischen Mathematiker Jan Lukasiewicz stammt der Vorschlag, bei arithmetischen Verknüpfungen den Operator nicht zwischen, sondern vor die Operanden zu setzen (daher auch „Präfix"-Schreibweise, z. B. +xy statt wie üblich x+y). Die polnische Notation kommt der Arbeitsweise des Computers entgegen, weil dort auch zuerst der Operator (+) und dann erst die Zahlenwerte verlangt werden. Das beschleunigt die Fehlerkontrolle und die Abwicklung der Arithmetik: Nach der Decodierung des Operationsteils ‚+' weiß die CPU, daß zwei Werte folgen, die zu addieren sind.

Verwandt mit dieser Schreibweise ist die von FORTH bekannte umgekehrte polnische oder „Postfix"-Notation. Dabei steht der Operator hinter den Operanden.

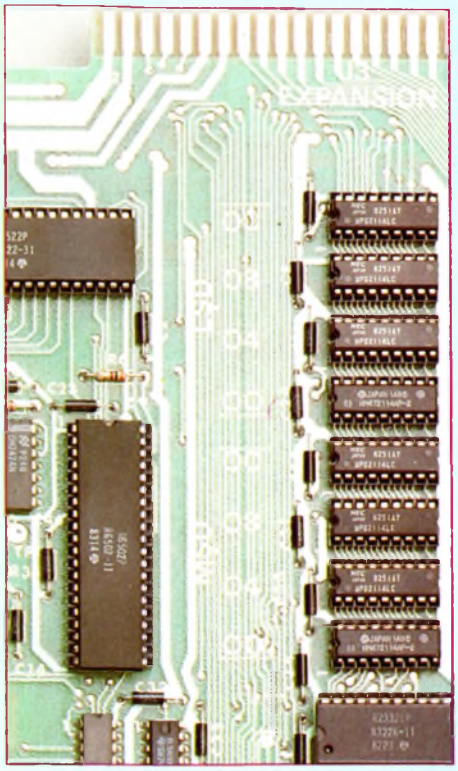
Bildnachweise
2101 Tony Sleep
2102, 2106, 2127 Caroline Clayton
2103, 2104, 2118, 2119, 2123 Kevin Jones
2120, 2121, 2125 Liz Heany
U 3 Chris Stevens

Polynomial = Polynom
Ein Polynom ist eine mathematische Funktion $y=f(x)$ in Gestalt einer Summe von Potenzen der Variablen x, die jeweils noch mit einer Konstanten multipliziert sind, z. B.
$y=5x^3 + 2x^2 + 2x + 8$
Polynomdarstellungen spielen in der numerischen Mathematik und daher auch in der Datenverarbeitung eine wesentliche Rolle, weil sich damit andere Funktionen in beliebig genauer Näherung berechnen lassen.

Port = Anschlußstelle
Die Peripherieanschlüsse eines Rechners für die Ein- oder Ausgabe von Daten werden auch als Port bezeichnet. Die externen Ports hängen im allgemeinen nicht unmittelbar am Prozessor-Chip, sondern es sind spezielle Schnittstellen-Bausteine zur Signalumsetzung zwischengeschaltet. Diese laufen unter Namen wie „PIA" (Peripheral Interface Adaptor = Peripherieschnittstellen-Anpassung) und vermitteln zwischen der CPU und den Peripheriegeräten etwa durch Parallel/Serien-Wandlung oder durch Baudratenangleich bei der Übertragung von Daten.

Printed Circuit =
Gedruckte Schaltung
Eine ‚gedruckte' Schaltung ist das Leiterbahnensystem zur Herstellung elektrischer Verbindungen zwischen einzelnen Bauelementen auf einer Isolierstoff-Trägerplatte. Dabei sind die Leiterbahnen aber nicht aufgedruckt, sondern aus einer flächigen Kupferkaschierung mit lichtempfindlicher Beschichtung herausgeätzt, nachdem das Leiterbahnenbild darauf optisch übertragen, entwickelt und fixiert wurde. In die Platine lassen sich Löcher für die ‚Pins' der Bauelemente bohren, die anschließend durch Bestückungsmaschinen automatisch aufgesteckt und montiert werden können. Bei doppelseitigen Platinen werden auf Vorder- und Rückseite Leiterbahnen angeordnet und punktweise „durchkontaktiert", das heißt leitend verbunden.

Ohne gedruckte Schaltungen wäre die Massenproduktion von Geräten wie Transistorradios oder auch Computern wohl kaum denkbar. Bei neueren Entwicklungen werden sog. ‚Multilayer'-Platinen eingesetzt, die durch Verkleben mehrerer dünner einfacher Leiterplatten und Durchkontaktieren realisiert wurden.



Die technologische Entwicklung bei den gedruckten Schaltungen führt zu immer feineren Leiterbahnen, bis in den Zehntelmillimeterbereich. Die Umsetzung von Schaltplänen in Leiterbahnenmuster (‚Entflechtung') erfolgt heute durchweg mit CAD-Verfahren am Bildschirm.

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Query Language = Datenbanksprache
Als „Query Language" wird die Kommandosprache eines Datenbank-Betriebssystems bezeichnet. Den eigentlichen Zugriff auf die Datenbasis organisiert das System anhand der vorgegebenen Suchparameter selbst. Häufig läßt die Query Language als „interaktive" Datenbanksprache auch einen Dialog mit dem Datenbanksystem zu.

Das Vokabular derartiger Sprachen besteht aus einer begrenzten Anzahl von Befehlen, die vom Rechner interpretiert und ausgeführt werden. Dem Benutzer bleibt dabei verborgen, wie die Datenbasis im einzelnen abgefragt wird. Bei anspruchsvolleren Datenbanksprachen ist meist nicht nur das Aufspüren einer einzigen Information und ihre Wiedergabe auf dem Bildschirm vorgesehen, sondern über bedingte Anweisungen läßt sich die Recherche beliebig ausdehnen.

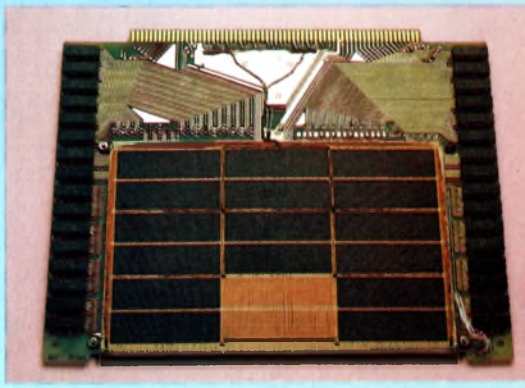
Queue = Warteschlange
Das Prinzip der Warteschlange ist etwa beim FIFO-Speicher realisiert, bei dem die zuerst eingegebenen Daten auch als erste wieder auftauchen, im Gegensatz zum „Stack" oder LIFO-Speicher.

FIFO-Speicher werden häufig als Buffer eingesetzt, vor allem bei Druckern. Da die Übertragungsrate des Rechners sehr viel höher ist als die maximale Druckgeschwindigkeit, werden die Zeichen zunächst in eine Warteschlange gestellt, die dann von vorn abgearbeitet wird.

Quicksort = Quicksort
Der „Quicksort" ist ein schneller Sortieralgorithmus. Dabei wird zunächst ein Mittelelement der ungeordneten Liste zur Vergleichsgröße erklärt und das Feld dann so in zwei Untermengen geteilt, daß alle Elemente der einen größer und alle der andern kleiner als das Vergleichselement sind. Für jede Teilmenge wird dann wieder eine Vergleichsgröße aus der Mitte genommen, abermals sortiert usw., bis die Teillisten nur noch ein Element enthalten – ihre Kettung ergibt das geordnete Feld.

Radix = Basis
In jedem Zahlensystem wird eine definierte Anzahl verschiedener Ziffernwerte benötigt – im Dezimalsystem sind es (mit der Null) zehn, im Hexadezimalsystem 16. Diese Zahl ist die „Basis" oder auch „Wurzel" des jeweiligen Systems. Bei mehrstelligen Zahlen bezeichnen die einzelnen Ziffern Vielfache von Potenzen der Basis; man spricht dabei auch von „Radix-Schreibweise". Im Zehnersystem gilt z. B.
$12{,}5 = 1*10^1+2*10^0+5*10^{-1} = 1*10+2*1+5/10$
Das Komma (oder der Punkt, engl. „Radix Point") trennt den ganzzahligen Teil vom gebrochenen, und zwar nicht nur im Dezimalsystem: Hexadezimal wäre z. B.
$23.FF = 2*16^1+3*16^0+15/16+15/16^2$
woraus sich dezimal 35,996 errechnet, und im Dualsystem entspräche 1000101.01 dezimal dem Wert 69,25.

Unter „Radix-Sorting" versteht man ein stellenweises Sortieren: Die Elemente einer numerischen Liste werden zuerst nach dem Ziffernwert der letzten Stelle geordnet, dann nach dem der nächsten usw. bis zur höchsten Stelle. Der „Radix Exchange" (stellenweiser Tausch) verläuft umgekehrt: Die Listenelemente werden nach der höchstwertigen Stelle in Teillisten einsortiert, innerhalb derer nun nach der nächstniedrigen Stelle geordnet wird usw. bis zur letzten Stelle.



Die Fortschritte in der Halbleitertechnologie haben zu einer drastischen Reduktion der Kosten und des Platzbedarfs für das einzelne RAM-Byte geführt. Die abgebildete Platine aus einem 1976 gebauten Minicomputer der Firma Computer Automation Inc. beherbergt einen Ferritkernspeicher mit etwa 2 K Kapazität – auf der gleichen Leiterplatte ließen sich heute 64 RAM-Chips zu je 256 K unterbringen.

RAM = RAM
Das „Random-Access Memory" (Direktzugriffs-Speicher) ist fast die wichtigste Komponente eines Rechnersystems. RAM-Bausteine enthalten in Gestalt von Halbleiterschaltungen zahllose Speicherzellen, die jeweils eine Null oder Eins aufnehmen können. Die Zellen sind i. a. in Matrixform angeordnet und werden von der CPU in Achtergruppen als Bytes adressiert; ihr Inhalt läßt sich auslesen, löschen oder überschreiben.

Es gibt im wesentlichen zwei RAM-Typen: Bei den „dynamischen" RAMs wird die Information auf winzigen Kondensatoren als elektrische Ladung gespeichert, die wegen der Leckströme alle paar Millisekunden „aufgefrischt" werden muß. Bei den „statischen" RAMs dienen als Speicherzellen dagegen Flipflops, die den logischen Zustand Eins oder Null ohne Auffrischung beliebig lange festhalten und außerdem einen schnelleren Zugriff gestatten.

Bildnachweise
2129, 2141, 2142,
U3: Marcus Wilson-Smith
2130, 2132, 2138, 2139, 2146, 2147, 2149, 2152, 2153, 2155: Caroline Clayton
2134: Liz Heany
2136, 2137, 2142: Kevin Jones

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Read/Write Head = Schreib/Lese-Kopf
Dieses Bauteil im Cassetten- und Diskettenlaufwerk besorgt beim Schreiben die oberflächliche Magnetisierung des Datenträgers und fragt sie beim Lesen wieder ab. Bei der Aufzeichnung werden Ströme durch eine kleine Spule mit Eisenkern im Magnetkopf geschickt, um entlang von „Spuren" in der ferromagnetischen Beschichtung die Bits „einzumagnetisieren". Das Lesen setzt voraus, daß sich das Speichermedium relativ zum Kopf bewegt: Dann wirkt auf die Kopfspule ein veränderliches Magnetfeld, das in der Wicklung dem Bitmuster entsprechend elektrische Spannungsimpulse induziert, die verstärkt und an die CPU übertragen werden.

Real Time = Echtzeit
Bei Echtzeit-Betrieb muß der Rechner auf jede Eingabe ohne Verzögerung reagieren. Bei einer Prozeßsteuerung ist das ebenso wichtig wie bei einem Flugsimulator, wo die Situation so realistisch wie möglich wirken soll.

Es kostet erhebliche Mühe, die Reaktionszeit bei solchen Systemen auf ein Minimum zu reduzieren. Welche Verzögerung noch zulässig ist, hängt weitgehend vom konkreten Anwendungsfall ab. Weder bei der Bestätigung einer Buchung am Flugbüro noch bei einem Kassenautomaten kommt es auf Millisekunden an, anders als etwa bei der radargestützten Flugüberwachung. Ein Echtzeit-System im eigentlichen Sinn verlangt extrem schnelle und effiziente Soft- und Hardware.

Record = Datensatz
Wie vieles andere hat auch „Record" in der Datenverarbeitung mehrere (allerdings verwandte) Bedeutungen. Meist ist ein „Record" eine Informationseinheit innerhalb eines „Files", die selbst wieder aus inhaltlich zusammengehörigen Datenfeldern aufgebaut ist. Dafür ist auch die Bezeichnung „logischer Datensatz" gebräuchlich.

Zu unterscheiden davon sind Records oder „Blöcke", aus denen die Information auf dem Datenträger hardwaremäßig zusammengesetzt ist. Diese „physischen" Datensätze stellen die kleinste Datenmenge dar, die der Magnetspeicher akzeptiert oder selbst an die CPU liefert. Bei CP/M etwa entspricht die Blocklänge mit 128 Byte gerade einem Diskettensektor. Ein physischer Datensatz bildet demnach logischerweise eine durch die Formatierung des Datenträgers definierte Einheit.

Recoverable Error = Regenerierbarer Fehler
Wenn der Rechner Daten vom Band einliest, stellt die Fehlererkennungsroutine (meist anhand einer Paritäts- oder Prüfsummenabfrage) hin und wieder Unstimmigkeiten fest. Oft handelt es sich dabei nur um ein einziges falsches Bit in einer langen Datei. Statt nun alles erneut einlesen zu müssen, gibt es eine Methode, bei der nur die fehlerhaften Daten neu geladen werden müssen. Gelingt das ohne abermaligen Protest, handelte es sich um einen „regenerierbaren" Fehler.

Großrechner und Minicomputer, bei denen die CPU direkt den Bandlauf überwacht, gehen meist von sich aus an die Behebung derartiger Fehler: Das Band wird nach dem Erkennen sofort gestoppt.

Zur Gewährleistung der nötigen Wirklichkeitstreue müssen bei einem Flugsimulator für das Pilotentraining alle Steuerungsreaktionen und Instrumenten-Rückmeldungen genauso spontan wie im realen Cockpit erfolgen. Um die erforderliche Ansprechgeschwindigkeit des Rechnersystems bei solchen Echtzeit-Problemen zu erzielen, kommt nur die Programmierung in einer Compilersprache in Frage.

Bildnachweise
2157,2158,2159: Mike Clowes
2176,2177,2178: Crispin Thomas
2161, 2180: Caroline Clayton
2182: Don McPhee

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Shell Sort = Shell Sort
Dies Sortierverfahren wurde 1959 von dem holländischen Mathematiker Donald Shell entwickelt. Beim Shell Sort erfolgen Vergleiche und Vertauschungen zwischen den Listenelementen weiträumiger als etwa beim Bubble Sort.

Wenn mit dem Shell Sort beispielsweise ein Feld A mit acht Zahlen – etwas A=7,5,3,8,1,6,4,2 – geordnet werden soll, werden zunächst Zweiergruppen aus dem 1. und 5., dem 2. und 6. ... Element betrachtet, also (7,1), (5,6), (3,4), (8,2). Wenn in einem solchen Paar die größere Zahl vorn steht, erfolgt ein Platztausch im ursprünglichen Feld A, womit sich eine neue Liste B=1,5,3,2,7,6,4,8 ergibt. Darin werden nun Vierergruppen durch Verketten des 1., 3., 5. und 7. sowie des 2.,4. ....Elements erzeugt, also (1,3,7,4),(5,2,6,8). Wenn in einer Kombination die Reihenfolge nicht stimmt (d. h. bei 7,4 und 5,2), werden in der Liste B die Plätze gewechselt, die dadurch in die Form C=1,2,3,5,4,6,7,8 übergeht. Nun sind nur noch die Zahlen 5 und 4 zu vertauschen.

Shift Register = Schieberegister
In einem solchen Rechner-Register lassen sich Binärzahlen Stelle für Stelle nach links oder rechts verschieben, so etwa für die Seriell/Parallel-Wandlung: Die nacheinander übertragenen Bits füllen sukzessiv ein Schieberegister auf, das dann parallel ausgelesen werden kann. Die entgegengerichtete Umsetzung kann durch bitweises Leerschieben des geladenen Registers erfolgen. Das alles besorgt die Hardware allein; der Programmierer kommt erst mit Schieberegistern in Berührung, wenn er in Assembler- oder Maschinencode bestimmte Speicherinhalte durch Bitmanipulation verändern will.

Dabei bedeutet das Rechtsversetzen einer Binärzahl um eine Stelle eine Division durch zwei, das Linksverschieben (unter Anfügen einer Null) die Multiplikation mit zwei. Die Zweierkomplement-Darstellung einer negativen Zahl bleibt beim Rechtsschieben jedoch nur erhalten, wenn links eine Eins nachgefüllt wird. Die „arithmetischen" Schiebebefehle der CPU bewirken dies automatisch, so daß beim Rechtsschieben aus einer −16 wirklich eine −8 entsteht; beim „logischen" Schieben („Rotieren") wird das Bitmuster dagegen durch Nachziehen der am einen Ende herausgefallenen Stelle ergänzt.

Simulation = Simulation
Simulationsprogramme dienen dazu, reale Vorgänge auf einem Computer nachzubilden. Ein Beispiel wäre ein Simulationsprogramm für die aerodynamische Optimierung einer Pkw-Karosserie, das den Bau von Modellen und Windkanalmessungen weitgehend erübrigt.

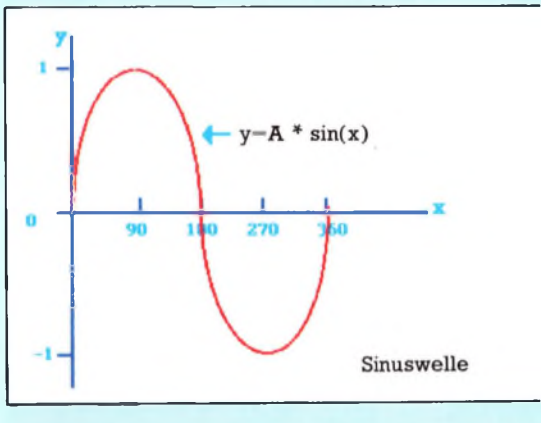
Eine weitere wichtige Anwendung sind rechnergesteuerte Simulatoren für spezielle Ausbildungszwecke, beispielsweise das Pilotentraining. Daran können auch Anfänger ohne Gefahr für teure Maschinerie oder gar Leib und Leben ihre Erfahrungen sammeln – selbst grobe Bedienungsfehler kosten nicht gleich Millionen oder den Kragen. Außerdem wird weder Kerosin vergeudet noch ein Flugzeug blockiert.

Simulationen ersparen aber nur dann Geld, wenn sie den Anforderungen der Wirklichkeit gerecht werden; für die Aussagekraft ist die exakte Nachbildung der Realität von größter Bedeutung. Bei der Simulationsplanung muß daher sorgfältig jeder in der Praxis relevante Faktor berücksichtigt werden.

Flugsimulator-Programme gibt es auch für eine Reihe von Heimcomputern; dabei muß der „Pilot" mit dem Joystick oder den Keyboard-Tasten manövrieren.

Sine Wave = Sinuswelle
Eine Sinuswelle wird mathematisch durch die Formel y=A * sin(x) beschrieben; der Sinus ist eine Winkelfunktion. Wenn Sie die Variable x kontinuierlich wachsen lassen, ergibt sich die unten wiedergegebene Kurve, die in charakteristischer Weise mit der „Amplitude" A um die x-Achse schwingt.

Sinuswellen werden oft benutzt, um das Übertragungsverhalten von Elektronik-Komponenten zu erfassen.



Sinuswelle

Bildnachweise
2186: Tony Sleep
2194: BBC Hulton PL, IBM
2208: Adrian Morgan
2192, 2195, 2203, 2205, 2210, 2211, 2212, U3: Caroline Clayton
2187, 2188, 2189: Mike Clowes
2200, 2201, 2206: Kevin Jones
2197, 2198, 2199: Chris Stevens

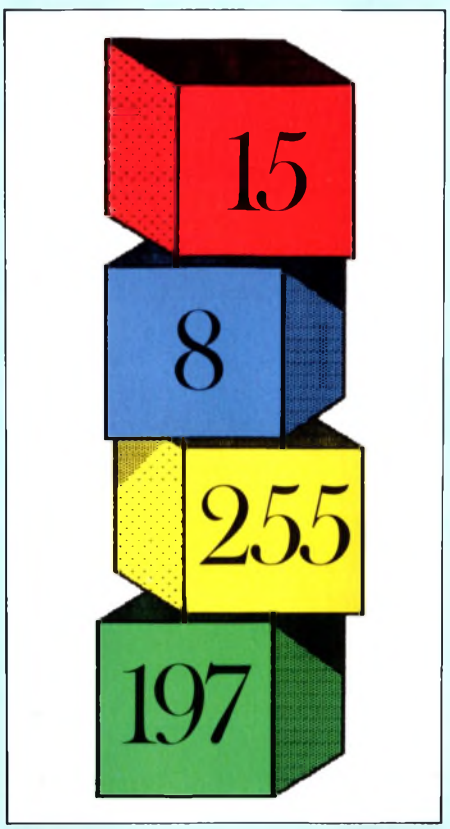
Fachwörter von A bis Z

Source Code = Quellprogramm
Ein Quellprogramm ist in einer Compiler- oder Assemblersprache geschrieben. Daraus muß vor Ablauf im Rechner durch Übersetzen in den Maschinencode ein „Objektprogramm" erzeugt werden. Das Quellprogramm, z. B. in PASCAL oder COBOL, wird zunächst einer syntaktischen Prüfung unterzogen; treten dabei keine Fehler auf, folgt die Compilierung. Auch ein Assemblerprogramm mit seinen mnemotechnischen Befehlen und symbolischen Adressen stellt ein Quellprogramm dar, das durch Assemblieren in die Maschinensprache umgesetzt wird.

Split Screen = Fenstertechnik
Bei dieser Technik wird der Bildschirm in zwei oder mehr Felder unterteilt, so daß oben auf dem Schirm ein Menü stehen kann, während darunter der Text dargestellt wird. Die untere Schirmhälfte läßt sich dabei unabhängig von der oberen bearbeiten, und der Benutzer kann auch meist beliebig zwischen den Fenstern hin und her springen. Das ist beispielsweise beim Editieren langer Schriftstücke sehr praktisch, weil ältere Passagen parallel zum Neuentwurf überarbeitet werden können. Besonders bei Abenteuerspielen finden Sie jetzt immer häufiger auch einen geteilten Grafikschirm.

Spooler = Spooler
Systemprogramme, die den Datenfluß an Peripheriegeräte (meist Drucker oder externe Speicher) auffangen und nach dem Warteschlangenprinzip in eigener Regie verwalten, heißen „Spooler". Sie sind vor allem deshalb zweckmäßig, weil die Ausgabegeschwindigkeit des Prozessors sehr viel größer ist als die Aufnahmefähigkeit der Peripherie. Wenn Daten dann zunächst in einen Puffer gesteckt werden, braucht die CPU nicht auf das externe Gerät zu warten. Zumeist arbeitet der Spooler unabhängig von der CPU und läßt sich direkt vom zugehörigen Peripheriegerät mitteilen, wann wieder neue Daten gesendet werden können.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



„Stacks" oder Stapelspeicher spielen in der Datenverarbeitung für das kurzfristige Deponieren von Informationen eine wichtige Rolle. Sie sind besonders einfach zu handhaben. Beim LIFO-Stack ist wie beim Würfelstapel aus dem Baukasten der Zugriff nur von oben her möglich. Die Reihenfolge der gespeicherten Größen bleibt vom Eintragen bis zum Auslesen unverändert, was für die Verwaltung der Rücksprungadressen bei verschachtelten Unterprogrammaufrufen ideal ist. Eine einfache Einsatzmethode für nicht so routinierte Anwender.

Square Wave = Rechtecksignal
Ein elektrisches Signal, das periodisch zwischen zwei Spannungspegeln hin- und herspringt, wird als „Rechtecksignal" bezeichnet. Eine solche Impulsfolge ist als Muster von binären Nullen und Einsen interpretierbar und dient zur Informationsübertragung, beispielsweise zwischen CPU und der Peripherie.

Stack = Stapelspeicher
Der „Stack" ist als spezieller Speicherbereich der CPU eines Computers zugeordnet und wird zur kurzfristigen Ablage von Registerinhalten und anderen Informationen benutzt. Die Kapazität des Stack und seine Lage im Arbeitsspeicher hängt vom jeweiligen Prozessor ab: Beim 6502 liegt der Stack auf der Nullseite (Zero Page) und faßt nur 256 Byte, beim Z80 dagegen können ihm beliebige Adressen und bis zu 64 KByte zugeteilt sein. Der Stack ist aber stets in gleicher Weise organisiert und geordnet.

Jeder Prozessor verfügt über besondere Stack-Befehle zum Eintragen (PUSH) und Entnehmen (POP) von Information. Beim üblichen LIFO-Stack werden die Daten „aufeinander" gestapelt; die letzten Einträge werden zuerst ausgelesen.

An diesem Prinzip kommt man beim Umgang mit dem Stack kaum vorbei: Die tieferstehende Information ist nur zugänglich, wenn der Stack von oben her weit genug abgearbeitet ist. Für den Zugriff hält ein bestimmtes CPU-Register, der „Stack Pointer" (Stapelzeiger), stets als aktuelle Adresse die der zuletzt angesprochenen Speicherzelle bereit. Der Stapelzeiger wandert beim Füllen und Leeren des Stack automatisch auf- bzw. abwärts, kann aber seitens des Programmierers bei Bedarf auch verschoben werden. Stack-Manipulationen sind nur im Maschinencode vorgesehen, aber es gibt sie auch in FORTH.

Bildnachweise
2213: Kevin Jones
2214: Liz Heany
2215–2217, 2240, U3: Caroline Clayton

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Thermal Printer = Thermodrucker
Der Druckknopf eines Thermodrukkers enthält eine Matrixanordnung von kleinen Heizwiderständen, die das Schriftbild durch Wärmeeinwirkung auf ‚Thermopapier' erzeugen. Dessen wachsartige Beschichtung wird beim Schmelzen transparent und läßt die darunter befindliche Farbe sichtbar werden.

Thermodrucker finden im Heimcomputerbereich Einsatz, da sie preiswert, sehr leise und ziemlich schnell sind.

Threaded List = Verknüpfte Liste
Zusatzinformationen an Datenfeldern sind nützliche Maßnahmen bei Datenbanksystemen oder Sprachkonzepten für die ‚Künstliche Intelligenz', so daß der Rechner über diese ‚Fäden' (threads) dann Querverbindungen folgen kann. Aus einer derart präparierten Liste läßt sich leicht eine bestimmte Auswahl von Datensätzen zusammenstellen.

Time Sharing = Timesharing
Dieses Verfahren für die Zuteilung von CPU-Zeit bei Mehrplatzsystemen arbeitet mit festen Zeitanteilen (Time Slices = ‚Zeitscheiben') im Millisekundenbereich, die den einzelnen Teilnehmern im Wechsel zugewiesen werden. Bei anderen Multiprogramming-Techniken werden die Zeitscheiben dagegen unter Berücksichtigung diverser Gesichtspunkte individuell bemessen.

Der Timesharing-Betrieb einer größeren Anlage ist kostengünstiger als das Arbeiten mit vielen kleinen Einzelsystemen, da die Auslastung von CPU und Peripherie besser ist. Ein Nachteil besteht darin, daß gleich eine sehr aufwendige Maschine angeschafft werden muß, die eine Vielzahl von Terminals bedienen kann und dabei so schnell ist, daß für die Benutzer keine störenden Wartezeiten in Erscheinung treten.

Top-Down Development = Top-Down-Programmierung
Bei diesem Vorgehen wird ein stufenweise immer feiner gegliederter Entwurf erstellt – es entsteht eine ‚Baumstruktur'. Erst wenn die Verästelung zu einfachen Teilaufgaben geführt hat, beginnt die Codierung.

Wenn Sie ein Programm schreiben wollen, das einen Roboter eine Tüte Milch aus dem Laden um die Ecke holen läßt, besteht die Gesamtaufgabe aus ‚Hingehen, Einkaufen, Zurückkommen'. Jeder dieser drei Aufträge wäre nun weiter zu unterteilen, das ‚Hingehen' z. B. in ‚Haus verlassen, Straße entlang, links, Geschäft betreten'. Als nächstes müßten Sie den Punkt ‚Haus verlassen' in ‚Tür ansteuern, Klinke fassen, . . .' zerlegen usw. Das Unterteilen der Aufgabenstellung in ihre Details führt schließlich zur Formulierung der Anweisungen für die Antriebsmotoren.

Trace Programm = Tracer
Ein Tracer oder ‚Ablaufverfolger' ist eine Hilfsroutine zum schrittweisen Protokollieren der Ausführung eines Programms, um die Fehlersuche zu unterstützen. Je nach Betriebssystem sind mehr oder weniger komfortable Tracer verfügbar, auch für Heimcomputer, wobei als Ein-/Ausbefehle häufig TRON (TRace ON) bzw. TROFF (TRace OFF) verwendet werden.

Die Ablaufprotokollierung erfolgt unterschiedlich detailliert, sie kann beispielsweise alle angesteuerten Zeilennummern enthalten oder nur die Prozedurnamen in der Reihenfolge ihres Aufrufs.

Thermodrucker erfreuen sich trotz einiger systembedingter Nachteile im Heimcomputerbereich großer Beliebtheit. Sie sind billig in der Anschaffung, leise und zuverlässig, brauchen aber teures Spezialpapier und bieten keine hohe Druckqualität.

Bildnachweise
2241: Claudia Zeff
2242: Frederic Voisin
2244, 2257, 2259, 2260: Caroline Clayton
2254, 2255, U3: Chris Stevens
2261: Liz Dixon
2261: Liz Heany
2263: Mike Brownlow

Fachwörter von A bis Z

UART = UART
Ein „UART" oder „Universal Asynchronous Receiver/Transmitter" ist eine asynchrone Universalschnittstelle mit einem Parallel/Seriell-Wandler und einem umgekehrt arbeitenden Umsetzer, beide in einem Chip vereinigt. UARTs dienen als Interface zwischen dem parallelen Rechnerbus und einem angeschlossenen seriellen Peripheriegerät oder Kommunikationsnetz.

Unconditional Branch = Unbedingter Sprung
Ein Sprungbefehl bewirkt allgemein, daß der Rechner aus der aktuellen Kommandofolge aussteigt und das Programm an einer anderen Stelle fortsetzt. Bei einem „unbedingten" Sprung ist die Ausführung nicht an das Erfülltsein einer oder mehrerer Bedingungen gebunden, sondern sie findet in jedem Fall statt.

Der Unterschied zwischen einem „bedingten" und einem unbedingten Sprung läßt sich z. B. an den entsprechenden 6502-Assemblerkommandos verdeutlichen: Die Anweisung BEQ = Branch (if) EQual (zero) ist ein bedingter Sprungbefehl – der Sprung erfolgt nur, wenn das Zero-Flag im Statusregister des Prozessors gesetzt ist. Das Kommando JMP = Jump veranlaßt dagegen einen unbedingten Sprung, d. h. die Fortsetzung der Programmbearbeitung bei der angegebenen Adresse ohne Abfrage einer Bedingung.

Unix = Unix
Das Betriebssystem Unix wurde in den Bell Laboratories zunächst für einen PDP-7-Minicomputer von DEC entwickelt und dann auf die PDP-11-Serie übertragen. Als Mehrbenutzersystem gestattet es den gleichzeitigen Zugriff mehrerer Teilnehmer auf dieselbe Anlage; es hat seit seiner Vorstellung im Jahr 1971 bei Minicomputern und 16-Bit-Micros große Verbreitung gefunden.

Unix war das erste Betriebssystem mit einer leistungsfähigen flexiblen Benutzerschnittstelle, und viel von seinem erfolgreichen Konzept – u. a. die hierarchische Dateistruktur – ist

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

bei anderen Systemen (z. B. bei MS-DOS) übernommen worden.

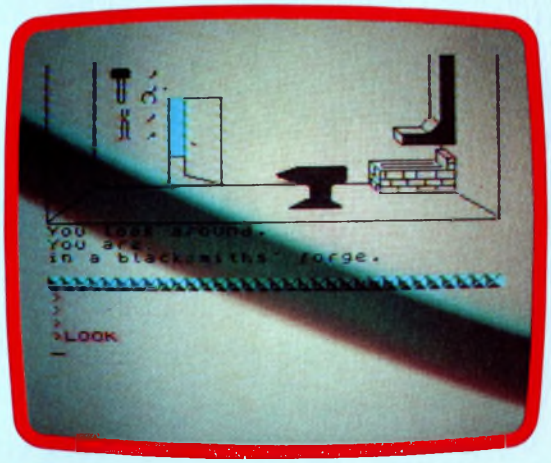
Validity Check = Gültigkeitskontrolle
Beim Validity Check wird softwaremäßig geprüft, ob die Eingangsdaten für ein Programm innerhalb des zulässigen Wertebereiches liegen. Vor allem bei der Tastatureingabe ist das sehr zweckmäßig. Die BASIC-Anweisung INPUT X z. B. erwartet das Eintippen von Ziffern und nicht von Buchstaben. Der erfahrene Programmierer baut in seine Routinen weiterreichende Gültigkeitskontrollen ein, um zu vermeiden, daß offensichtlich fehlerhafte Daten in die eigentliche Verarbeitung geraten und dann vielleicht einen Programmabsturz auslösen.

Der Validity Check erfolgt im Rahmen der Datenübergabe an ein Programm. Durch ein paar zusätzliche Anweisungszeilen läßt sich leicht feststellen, ob die Eingangsgrößen akzeptabel sind; wenn nicht, ergeht an das Eingabegerät die Aufforderung, neue Werte bereitzustellen.

Variable = Variable
Einzelne Zeichen oder Zeichenfolgen, die in höheren Programmiersprachen als Platzhalter für numerische oder alphanumerische Größen verwendet werden, bezeichnet man als „Variable"; im Laufe der Programmausführung lassen sich ihnen wechselnde Werte zuweisen. Variablen stellen wesentliche Elemente der Programmierung dar, weil sie neben der Einführung sinnvoller Namen auch den Aufbau von Datenstrukturen erlauben.

VDU = Datensichtgerät
VDU bedeutet „Video Display Unit", und dieser Begriff wird oft nicht nur für das Sichtgerät selbst gebraucht, sondern auch für die Elektronik-Baugruppe zur Erzeugung der Steuersignale im Rechner. Das Datensichtgerät arbeitet mit einer Kathodenstrahlröhre, auf deren Schirm das Bild zeilenweise durch einen Raster-Scan aufgebaut wird. Im Prinzip genügt daher als Sichtgerät auch schon ein Heimempfänger, obwohl spezielle Monitore mit RGB-Eingang bessere Bilder liefern. Bei tragbaren Computern setzen sich jetzt zunehmend Flachbildschirme mit reduziertem Leistungsbedarf durch, entweder als LCD-Anzeigen oder als Plasma- bzw. Luminiszenz-Displays.

Die Anzeige muß bei Sichtgeräten kontinuierlich aufgefrischt werden, was zwar einigen elektronischen Aufwand erfordert, aber andererseits die rasche Veränderung der Bilder auf dem Schirm zuläßt.



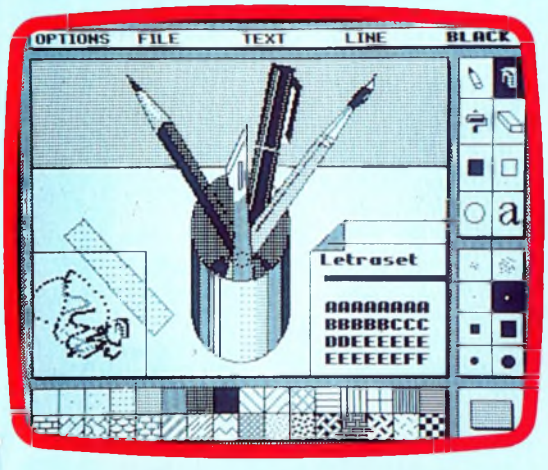
Bei den Datensichtgeräten mit Bildröhre wird der Schirm fünfzigmal pro Sekunde durch einen Elektronenstrahl abgetastet. Der Eindruck einer gleichmäßigen Bildhelligkeit entsteht wesentlich durch das Nachleuchten des Phosphors.

Bildnachweise
2270: Kevin Jones
2273, 2291, 2293: Caroline Clayton
2276–2289: Johanne Ryder

Fachwörter von A bis Z

WIMP = WIMP
WIMP bedeutet „Windows Icons Mouse Program", zu deutsch „maus- und Piktogramm-gesteuertes Fenstermenü-Programm". Die Bedienung erfolgt dabei nicht durch Eintippen von Kommandos, sondern über die Maus, indem ein Cursor-Symbol auf „Piktogramme" – die sinnbildlichen Darstellungen der Operationen – gesetzt wird.

Die WIMP-Technik besticht durch extreme Benutzerfreundlichkeit. Das Konzept geht auf Forschungsarbeiten der Firma Xerox zurück, die gegen Ende der siebziger Jahre zur Entwicklung der SMALL-TALK-Benutzerumgebung für das objektorientierte Programmieren führten. Den ersten kommerziellen WIMP-Systemen für Microcomputer stand die Fachwelt zuerst sehr skeptisch gegenüber, aber inzwischen ist bei der Mehrzahl der neuen Rechner diese Technik entweder bereits integriert oder nachträglich zu implementieren.



Die WIMP-Technik erregte mit der Vorstellung von Apples LISA erstmals Aufsehen und wurde dann durch den Macintosh allgemein bekannt.

Word = Wort
Ein „Wort" wird durch eine Gruppe zusammengehöriger Bits gebildet, die als Einheit adressierbar sind. Die Anzahl der Bits oder die „Wortlänge" ist generell durch die Verarbeitungsbreite der bei einem Rechner verwendeten CPU bestimmt; bei Micros beträgt sie normalerweise acht, 16 oder 32 Bits (d. h. ein, zwei oder vier Bytes).

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Word Processing = Textverarbeitung
Textverarbeitungssysteme erleichtern das Entwerfen von Schriftstükken und ihre Bearbeitung bis hin zum druckfertigen Formatieren des Textes. Daher sind neben den Editierfunktionen wie Einfügen, Löschen oder Umstellen von Textpassagen meist auch der automatische Zeilenumbruch und das Ausrichten der Zeilenränder vorgesehen.

Im privaten wie im kommerziellen Bereich ist die Textverarbeitung ein bevorzugtes Einsatzfeld für Microcomputer. Es gibt eine Riesenauswahl derartiger Systeme: von kompletten Spezialanlagen für einige zehntausend Mark über anspruchsvolle Softwarepakete für einbis zweitausend bis zum einfachen Texteditor für zwanzig Mark.

Workspace = Arbeitsbereich
Als „Workspace" reservieren sich viele Programmsysteme einen bestimmten Speicherbereich, in dem sie dann während des Ablaufs kurzfristig Daten sowie Variablenwerte ablegen.

Xerographic Printer = Xerografischer Drucker
Dieser Drucker hat eine mit einer Halbleiterschicht überzogene Platte, die elektrostatisch aufgeladen wird. Bei Belichtung ändert sich die Leitfähigkeit entsprechend der Bildstruktur, so daß sie mit Hilfe eines Pulvers auf Papier gebracht werden kann.

Zero Flag = Zero Flag
Das Statusregister der CPU enthält eine Reihe von Markierungs-Flipflops oder „Flags", die als Einzelbits bestimmte Zustände anzeigen. Das „Zero-Flag" wird immer dann gesetzt, wenn eine Akkumulatoroperation zum Ergebnis Null geführt hat.

Das Zero-Flag spielt in der Maschinenprogrammierung eine wichtige Rolle beim Vergleichen von Größen oder beim Überwachen von abwärtslaufenden Schleifenzählern. Die CPU fragt dann in Verbindung mit einem bedingten Sprungbefehl das Zero-Flag ab und verzweigt, wenn es gesetzt ist, zu einem anderen Programmteil.

Zero Page = Nullseite
Zur Angabe absoluter Adressen sind bei einem 8-Bit-Micro zwei Bytes erforderlich, nämlich ein höher- und ein niederwertiges. Das höherwertige Byte enthält die Nummer der „Seite" (Page), auf die sich die Adresse bezieht, das niederwertige legt die Position innerhalb der Seite fest. Jede Seite umfaßt 256 aufeinanderfolgende Speicherworte. Die erste Seite des Speichers wird als „Zero Page" bezeichnet, weil das höherwertige Adreßbyte dort Null ist. Zum Ansprechen der Nullseite genügt also eine Ein-Byte-Adresse – Befehle mit Nullseitenadressierung benötigen weniger Zugriffzeit als andere. Bei vielen Rechnern benutzt daher das Betriebssystem die Nullseite als Arbeitsbereich zur Ablage wichtiger Variablen und Adressen.

Als „Zero Page" gelten zwar eigentlich nur die ersten 256 Speicherbytes, aber einige Prozessoren kennen auch ein „Direct Page Register" (Seitenregister für die Direktadressierung) zum Eintragen einer beliebigen Seitennummer, auf die dann schneller zugegriffen werden kann.

Bildnachweise
2297: Tony Sleep
2298, 2310, 2318: Kevin Jones
2299, U3: Ian McKinnel
2303: Dimension Graphics
2304: Paul Bryant
2309: Liz Heany
2312, 2316, 2321: Caroline Clayton

Das Entlöten

Wer schon einmal mit Erfolg ein Anschlußkabel selbst gemacht hat, darf sich auch an Schwierigeres wagen.

In der folgenden Übung wird beschrieben, wie Sie mit etwas Geschick aus Ihrem ZX 81 eine Maschine machen, die neben BASIC auch FORTH versteht. Wie man richtig lötet, wurde bereits erklärt. Wie aber eine solche Verbindung sauber wieder lösen? Solange es dabei nur um Kabel oder Stecker geht, ist das kein Problem. Anders bei elektronischen Bauteilen: Selbst ein einfacher Transistor hat schon drei Anschlüsse (Pins). Um aber einen Chip von der Platine zu lösen, müssen sämtliche Pins – das können bei einigen Prozessoren bis zu 40 sein – vom Lot befreit werden. Es ist praktisch unmöglich, alle Anschlüsse gleichzeitig bis zum Schmelzpunkt des Lötzinns zu erhitzen. Statt dessen wird das Lot nach und nach von den einzelnen Pins entfernt.

FORTH-ROM statt BASIC

Natürlich kann man zur Übung einen beliebigen Chip aus- oder zusätzliche RAM-Chips zur Speichererweiterung einlöten. Unser Projekt hat jedoch gleichzeitig einen echten Nutzen für Sie: Das BASIC-ROM des ZX 81 soll entfernt und durch einen Sockel ersetzt werden. Dieser ist durch ein Flachkabel mit einem Veroboard außerhalb des Rechners verbunden. Auf dieses wird wiederum ein Sockel montiert. In ihn kann später nicht nur das ausgebaute BASIC-ROM sondern auch ein anderer Chip eingesetzt werden. Für diese Übung eignet sich der ZX 81 besonders gut, weil es für das Gerät die Sprache FORTH in ROM-Form gibt, das zusätzlich auch noch Multi-Tasking ermöglicht: Dadurch können mehrere Programme gleichzeitig, jedoch völlig unabhängig voneinander, ablaufen. Für diese respektable Leistung braucht der kleine ZX 81 allerdings mindestens 2 KByte RAM, muß also mit einem zusätzlichen Speicher-Modul aufgerüstet sein.

Die hier vorgeschlagene Aufgabe wird Sie sicherer im Umgang mit elektronischen Bauteilen machen – sie erfordert aber auch sauberes und genaues Arbeiten. Im nächsten Teil unseres Bastel-Kurses wird gezeigt, wie die Erweiterung mit einem Multimeter durchgeprüft werden kann. Mit dem Multimeter können Sie Spannung, Strom und Widerstand messen. Beim Test von Bauteilen und Stromkreisen ist es ein unentbehrliches Hilfsmittel.

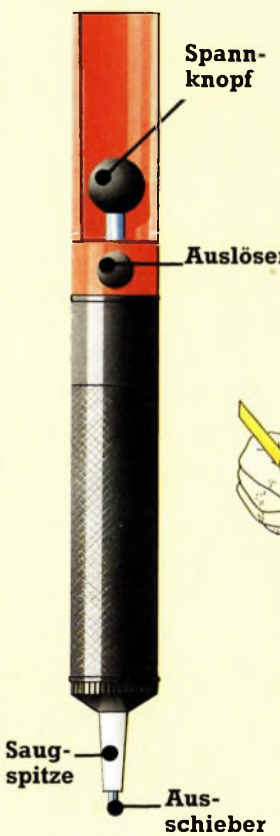
Zuletzt sollten alle Lötstellen noch einmal überprüft werden. Beim nächsten Mal zeigen wir, wie man mit einfachen Mitteln zum Ziel kommt, und den Umgang mit Meßgeräten.

Was Sie brauchen . . .

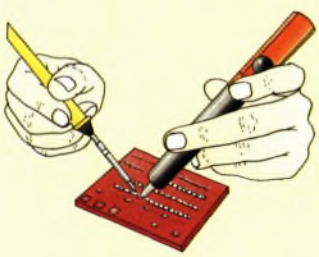
Wenn Chips in Ihrem Computer ausgetauscht, ersetzt oder durch zusätzliche Bauteile ergänzt werden sollen, empfiehlt sich der Einbau eines Chip-Sockels. Dadurch können integrierte Bauteile sehr schnell ausgewechselt werden. Neben den bereits beschriebenen Werkzeugen brauchen Sie außerdem eine Rolle Entlöt-Litze oder eine Lötsaugpumpe. In unserem Beispiel wird das BASIC-ROM des ZX 81 durch einen Standard-Sockel ersetzt, von dem aus ein Flachbandkabel zu einem Chip-Sockel außerhalb des Gehäuses führt. So kann der ZX 81 – je nach eingesetztem ROM – als BASIC- oder FORTH-Rechner verwendet werden. Außer dem Entlötgerät benötigen Sie noch eine kleine Lochplatine (Veroboard) und 28poliges Flachband-Kabel – alles Teile, die Sie in jedem Elektronik-Bastelgeschäft ohne Schwierigkeiten finden werden. Bei dem FORTH-ROM handelt es sich übrigens um eine Entwicklung von David Husband. Fragen Sie Ihren Sinclair-Händler danach.

Zuerst muß das Gehäuse des ZX 81 geöffnet und das BASIC-ROM gesucht werden. Von den insgesamt fünf Gehäuseschrauben sitzen drei unter den aufgeklebten Gummifüßchen am Boden des Rechners. Unter dem Gummifuß neben den EAR- und MIC-Anschlüssen befindet sich keine Schraube. Lösen Sie die Gummis vorsichtig, und drehen Sie alle fünf Kreuzschlitz-Schrauben los. Nun können Sie die Bodenplatte abnehmen, und die Unterseite der Platine kommt zum Vorschein. Nach Lösen der drei sichtbaren Kreuzschlitz-Schrauben kann die Platine aus dem Gehäuse herausgenommen werden. Zwei der Schrauben sitzen nahe beim Vielpol-Stecker, die dritte befindet sich neben dem Kühlkörper. Wenn Sie die Platine jetzt herumdrehen, sehen Sie das BASIC-ROM etwas links über dem Tastaturanschluß.

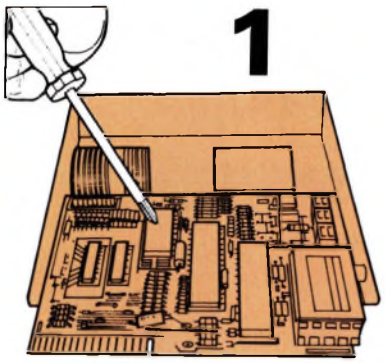
Werkzeug zum Entlöten

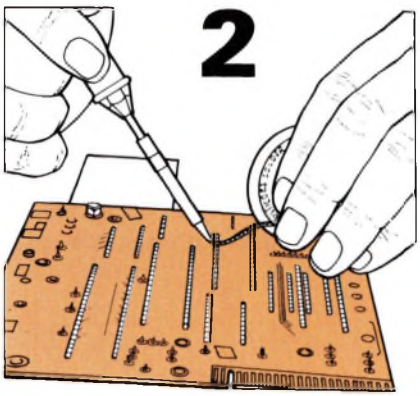


Beim Entlöten sind zwei verschiedene Techniken gebräuchlich: Wer nur selten Bauteile auswechselt, arbeitet mit der preiswerten Entlötlitze. Sie besteht aus feinsten Kupferdrähtchen, die mit Flußmittel überzogen sind. Durch den Kapillar-Effekt saugt die Litze flüssiges Lötzinn auf wie ein Lampendocht. Entlötlitze können Sie in unterschiedlichen Breiten und Längen kaufen. Das richtige Maß hängt von der Menge des Lötzinns ab, das entfernt werden soll. Ein Meter der nicht wiederverwendbaren Litze kostet ca. 4 Mark. Alternativ können Sie aber auch mit einer Lötsaug-Pumpe arbeiten, was auf die Dauer billiger ist. Das Gerät arbeitet nach dem umgekehrten Prinzip einer Fahrrad-Luftpumpe.



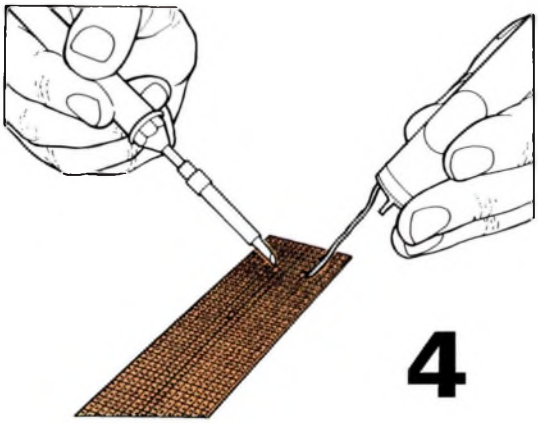
So wird entlötet: Lötpunkt mit dem Kolben bis zur Schmelze erhitzen, Feder der Lötsaugpumpe spannen und die Spitze des Gerätes am Lötpunkt ansetzen. Auslöseknopf drücken.





Wenn die Temperatur hoch genug ist, gibt es beim Entlöten keine Probleme. Einfach die Entlötlitze mit der Spitze des Kolbens auf die Lötstelle pressen, bis das Flußmittel der Litze schmilzt – dabei entsteht ein kleines Rauchwölkchen. Danach sollte die Litze das Lötzinn aufsaugen. Das vollgesogene Litzenende muß jeweils abgeschnitten werden.

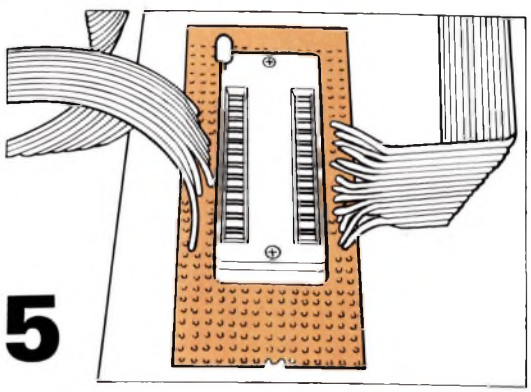
Flachkabel gibt es in unterschiedlicher Breite. Bei diesem Beispiel wird ein 28poliges gebraucht. Sie können natürlich auch zwei Kabel aus jeweils 14 isolierten Drähten verwenden. Alle 28 Enden werden 10 mm lang abisoliert und verzinnt. Auf der Chipseite mit dem halbkreisförmigen Ausschnitt anfangen und die Enden nacheinander sorgfältig festlöten. Durch den Ausschnitt am Chip können Sie erkennen, wie das Bauteil richtig eingesetzt ist. Gelegentlich findet man auch aufgedruckte Punkte als Hilfe.



Jetzt können Sie das Flachkabel auf die gewünschte Länge schneiden und auf der freien Seite ebenfalls verzinnen. Beim Anlöten der Enden auf der Nebenplatine muß jede Leitung genau an den der Hauptplatine entsprechenden Anschlußpunkt führen. Vorsicht, hier darf nichts verwechselt werden!

Lochplatinen (Veroboard)

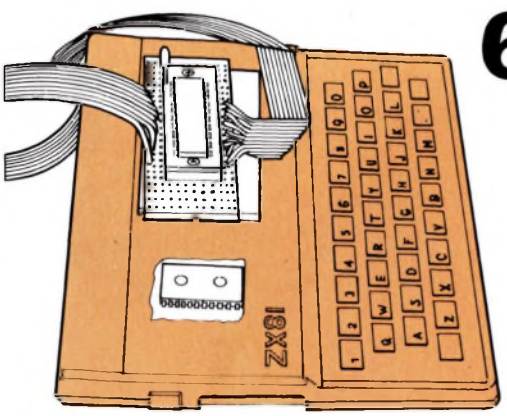
Vorgelochte Platinen mit kupfernen Leiterbahnen sind für den Selbstbau elektronischer Schaltungen gedacht. Die Leiterbahnen lassen sich je nach Bedarf durchschneiden oder -kratzen. Beim Einkauf sollten Sie darauf achten, daß das Rastermaß der Platine zum Rastermaß des geplanten Sockels bzw. ICs paßt.



Den Sockel lötet man am besten auf ein Stück Lochplatine (Veroboard), das man in geeigneter Größe abbricht oder absägt. Der Sockel wird so eingesteckt, daß jeder Lötstift auf einer eigenen Leiterbahn festgelötet werden kann. Zuerst zwei Eckpunkte festlöten, dann sitzt der Sockel beim Arbeiten fest.
Auf „kalte" Lötstellen achten!

Achtung!

Während der Garantiefrist darf Ihr Heimcomputer nur vom Hersteller bzw. der Vertragswerkstatt geöffnet werden. Meist wird nach selbständigen Eingriffen in das Gerät jede weitere Garantieleistung verweigert.



Wenn alles fertig gelötet ist, sollten Sie jede einzelne Verbindung noch einmal genau überprüfen. Das gilt für die Platine des Rechners genauso wie für die neue Tochterplatine. Ist Lötzinn zwischen zwei Anschlußpunkte gelaufen? Gibt es abstehende Drähtchen, die einen Kurzschluß erzeugen? In Zweifelsfällen mit der Lupe nachprüfen. Sicherheitshalber können Sie auch mit einer Nadel zwischen zwei Anschlußpunkten hindurchkratzen.

Kontakte prüfen

Letzter Schritt bei elektronischen Basteleien ist eine gründliche Prüfung der Lötstellen vor dem Einschalten des Gerätes. Ein einziger Kurzschluß kann empfindliche Bauteile schnell zerstören. Leitungsprüfer kosten weniger als 10 Mark, und sogar ein richtiges Vielfach-Meßgerät ist mit seinem Preis von 50 Mark immer billiger als manches versehentlich „hingerichtete" IC!

Wichtig ist zuerst einmal die Überprüfung der Lötstellen. Äußerlich kann auch eine bei zu niedriger Temperatur gelötete Verbindung akzeptabel aussehen – ohne daß sie tatsächlich Strom leitet. Mit vorsichtigem Ziehen läßt sich die Fehlerquelle häufig schon entlarven. – Je früher man sie findet, um so besser!

Wenn Sie alle Lötstellen auf diese Art geprüft haben, ist das Meßgerät an der Reihe. Das einfachste Gerät dieser Kategorie ist der Kontaktprüfer, wie man ihn im Werkzeug- oder Autozubehörgeschäft bekommt. Damit können Sie testen, ob zwei Kontaktpunkte miteinander leitend verbunden sind.

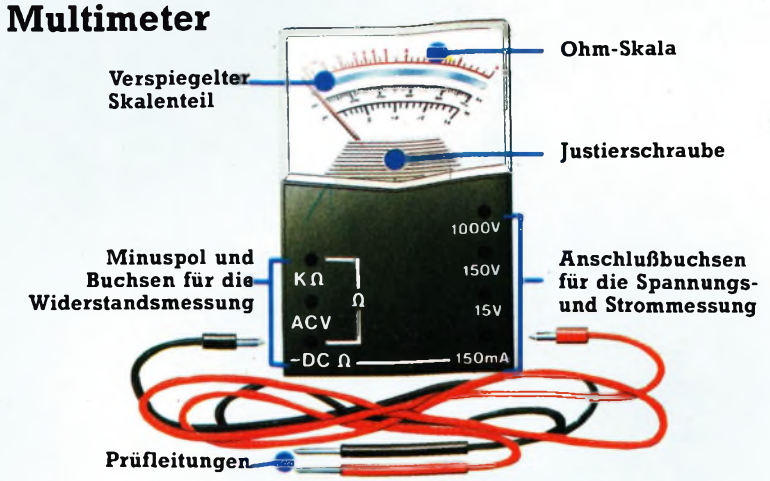
Für unsere Zwecke ist die Krokodilklemme, die sich am Kabelende des Spannungsprüfers befindet, nicht ganz das Richtige: Sie sollten besser eine Lötspitze hineinklemmen und das Ganze mit Isolierband umwickeln.

Ein Vielfachmeßgerät oder „Multimeter" ist schon ein etwas komfortableres Gerät: Damit läßt sich nicht nur auf Spannung testen, sondern man kann auch Widerstände messen. Einheit des elektrischen Widerstandes ist das „Ohm", so genannt nach dem Physiker Georg Ohm (1789–1854), der den Zusammenhang zwischen Strom, Widerstand und Spannung als erster beschrieb.

Eine solide Lötverbindung hat einen sehr geringen Widerstand: Das Anzeigelämpchen Ihres Leitungspüfers sollte also hell leuchten, der Zeiger des Multimeters muß ganz nach rechts ausschlagen. Ist das nicht der Fall, kann man auf eine „kalte" Lötstelle schließen, die noch einmal nachgearbeitet werden muß.

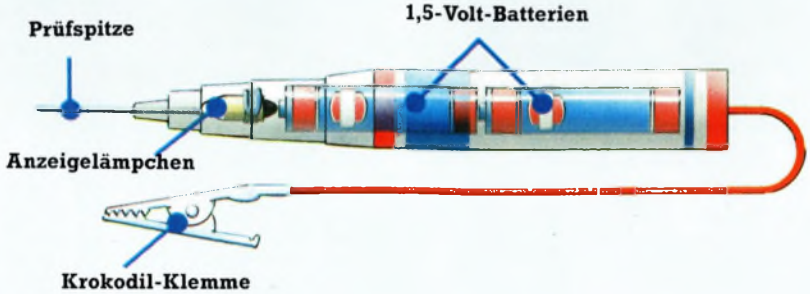
Mit dem Multimeter können Sie auch Spannung und Stromstärke messen: Liegt z. B. über einem 1-Ohm-Widerstand eine Spannung von 1 Volt, so fließt ein Strom von genau 1 Ampere.

Multimeter

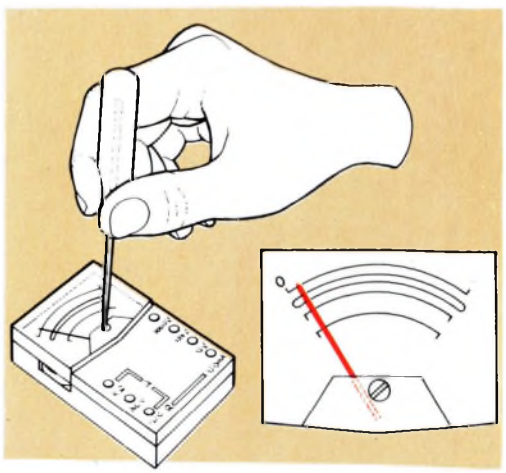


Die Skala eines Multimeters, mit dem man auf Durchgang und Widerstand prüfen kann, ist in „Ohm" geeicht. Zusätzliche Skalenteilungen gibt es meist für Strommessung (in Ampere) und für Spannungsmessungen, die auf der Volt-Skala abgelesen werden. Für ein Multimeter können Sie 50 Mark, aber auch 5000 Mark ausgeben – je nach erforderlicher Meßgenauigkeit. Die Anzeige erfolgt entweder digital, also mit Ziffern, oder analog, d. h. mit einem Zeiger über einer Skala. Die Analoginstrumente arbeiten mit einer Drehspule, an der der Zeiger befestigt ist. Sie sind meist preiswerter als die Digital-Multimeter.

Spannungsprüfer

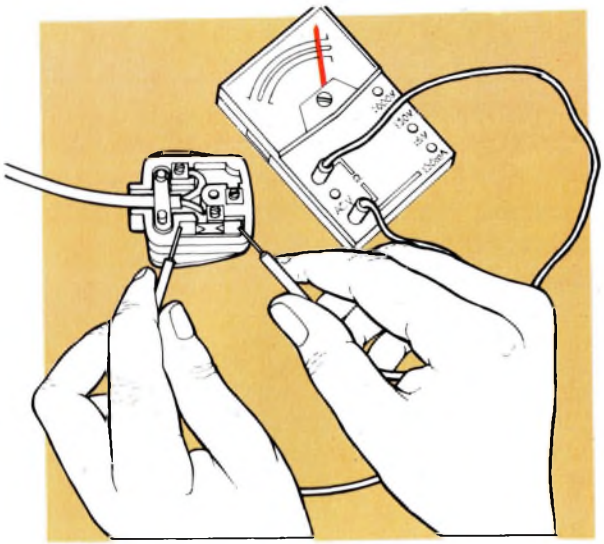


Nullpunkteinstellung



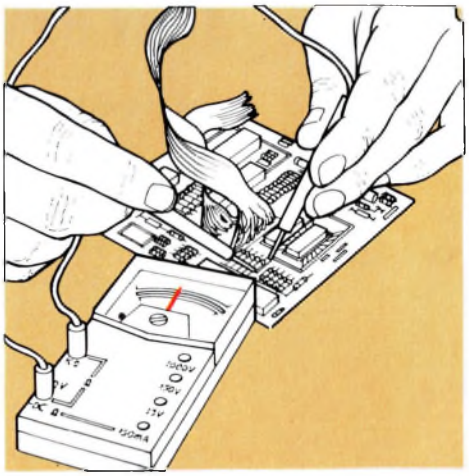
Analoge Meßgeräte, bei denen der Zeiger mechanisch bewegt wird, muß man vor Gebrauch justieren. Meist ist für diesen Zweck eine Schraube nahe beim Drehpunkt des Zeigers vorgesehen. Die seitliche Rändelschraube am Multimeter ermöglicht außerdem den Ausgleich der absinkenden Batteriespannung bei der Widerstandsmessung. Sie wird nachgestellt, wenn der Zeiger nicht mehr ganz nach rechts ausschlägt, wenn sich die Prüfspitzen berühren.

Spannung

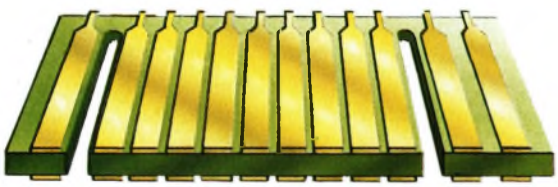


Eine der größten Fehlerquellen in elektrischen Geräten ist eine durchgebrannte Sicherung. Das passiert immer dann, wenn der Stromfluß den festgelegten Sicherungswert überschreitet. Natürlich kann man die Sicherung auf den bloßen Verdacht hin austauschen – besser ist, sie entweder mit dem Spannungsprüfer oder mit dem Multimeter zu testen. Wenn der Strom ausbleibt, die Sicherung aber in Ordnung ist, kann man mit Meßgeräten meist recht schnell die Fehlerursache ermitteln.

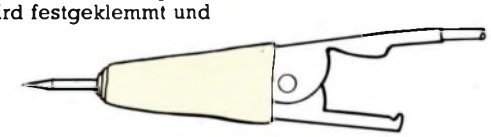
Widerstand



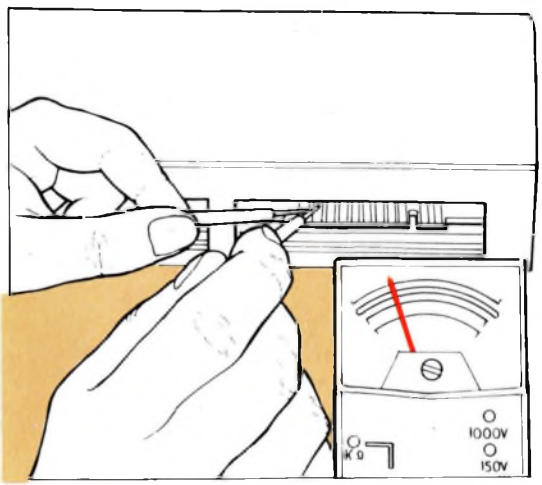
Steckplätze
Steckplätze der unten abgebildeten Art dienen zum Anschluß von Peripheriegeräten an den Computer. Manchmal ist nur ein einziger solcher Steckplatz vorhanden, etwa beim Sinclair Spectrum, oft gibt es aber auch mehrere dieser Schnittstellen. Diese Steckplätze sind in der Leiterplatine integriert.

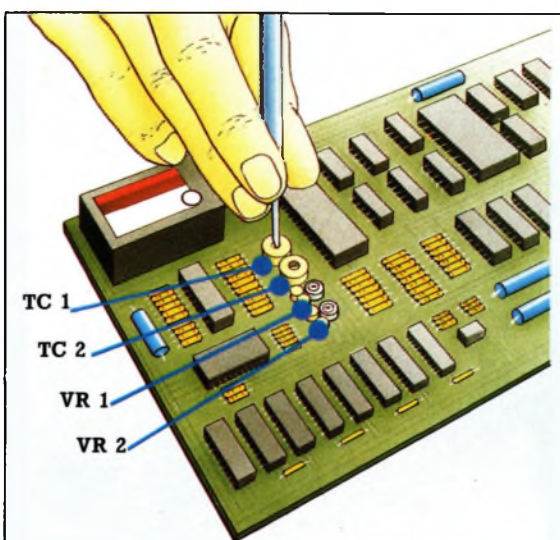


Krokodil-Klemmen
Die relativ groben Krokodil-Klemmen des Spannungsprüfers lassen sich leicht verfeinern: Eine dünne Lötkolbenspitze wird festgeklemmt und mit einem Isolierband-Streifen an der Klemme befestigt.



Spannungen messen
Ein gutes Beispiel für die Anwendung des Multimeters ist die Überprüfung der Betriebsspannung an der Platinen-Schnittstelle Ihres Rechners. Im Handbuch ist die Position der Leiterbahnen für 0 Volt, +5 Volt und +9 Volt angegeben. Die schwarze (negative) Prüfspitze wird auf die Bahn für 0 Volt, die rote Spitze auf die +5- oder +9-Volt-Bahn gehalten. Die Skala zeigt, ob die angegebenen Spannungswerte des Rechners korrekt sind.





Optimierung der Bildwiedergabe

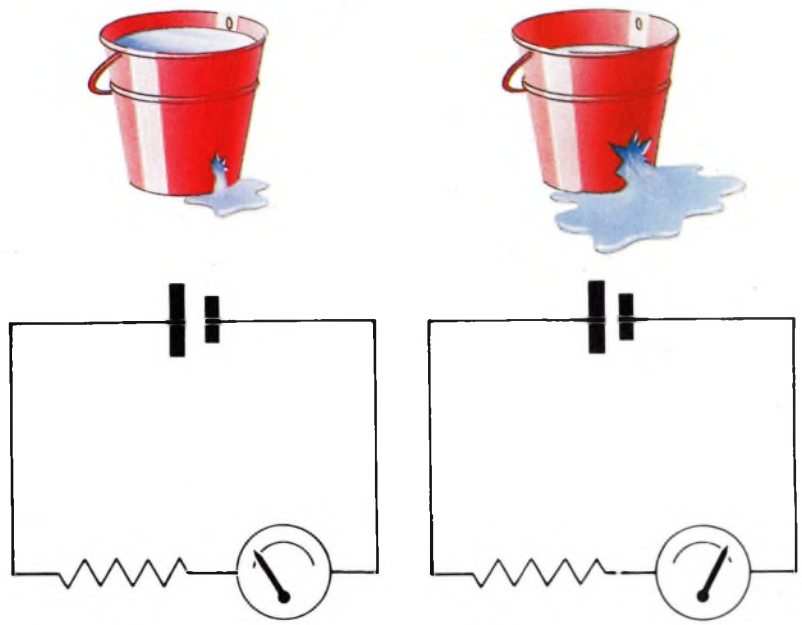
Wie bereits erwähnt, läßt sich bei den ersten Spectrum-Geräten die Qualität der Bildschirmdarstellung verbessern. Die Potentiometer VR 1 und VR 2 regeln die Farbbalance zwischen Rot/Grün und Blau/Gelb. Mit den Drehkondensatoren TC 1 und TC 2 wird die Zeichendarstellung und Farbsättigung beeinflußt. Das Gehäuse wird durch Lösen der fünf sichtbaren Kreuzschlitz-Schrauben an der Unterseite geöffnet. Denken Sie dabei aber an Ihre Garantie – sie verfällt beim Öffnen des Gehäuses! An Spectrum-Rechnern der Version 3 kann man allerdings keine Einstellarbeiten vornehmen.

Achtung!

Während der Garantiezeit darf Ihr Heimcomputer nur vom Hersteller bzw. seiner Vertragswerkstatt geöffnet werden. Nach Eingriffen in das Gerät wird meist jede weitere Garantieleistung verweigert.

Strom und Spannung

Elektrizität läßt sich am besten über ihre Basisgrößen Strom, Spannung und Widerstand mathematisch erklären.

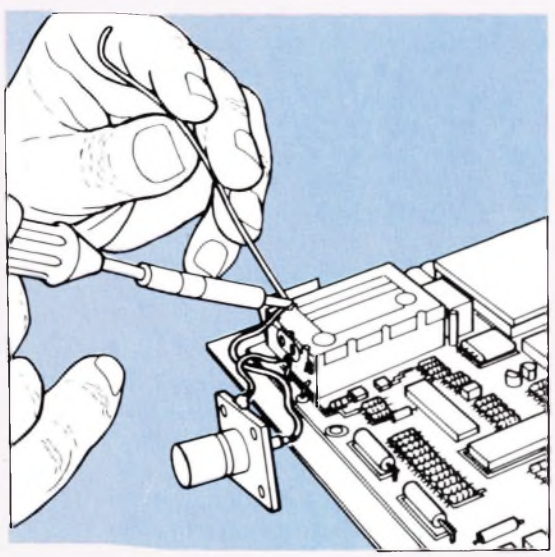


Strom und Widerstand

Die Bauteile in einem elektrischen Stromkreis setzen dem Stromfluß einen gewissen Widerstand entgegen, vergleichbar mit der Wirkung der Ventile und Rohre eines Heizungskreislaufes: Je größer der Widerstand ist, um so weniger Wasser fließt. Durch das Loch in einem Eimer läuft um so weniger Wasser, je kleiner das Loch ist. Wenn in einem Stromkreis ein hoher elektrischer Widerstand liegt, fließt nur ein geringer Strom. Verringert man den Widerstand, wird der Stromfluß größer. Der elektrische Widerstand eines Bauteils hängt hauptsächlich vom verwendeten Material ab – Leitungsdrähte etwa werden aus Kupfer hergestellt, weil es dem Strom nur geringen Widerstand entgegensetzt.

Klare Sicht

Nach einem einfachen Umbau kann der Sinclair Spectrum ein Videosignal liefern, mit dem sich ein hochauflösender Farbmonitor ansteuern läßt. Der Bild-Modulator ist als größerer silberner Kasten (oben links auf der Platine) leicht erkennbar. Am hinteren der beiden Anschlüsse des Modulators wird eine kurze Leitung angelötet, die Sie mit dem Außenkontakt einer BNC-Buchse verbinden. Den Mittelkontakt der Buchse schließen Sie an das Gehäuse des Modulators an.



In der letzten Folge unseres elektronischen Bastelkurses wurden einige Grundlagen der Elektrotechnik dargelegt. Doch in welcher Beziehung stehen die einzelnen Größen zueinander und wie kann man sie berechnen? Der deutsche Physiker Georg Ohm (1789–1854) hat als erster den Zusammenhang zwischen Strom, Spannung und Widerstand untersucht. Ohm erkannte, daß sich aus zweien dieser Größen jeweils die dritte errechnen läßt: Die Spannung ergibt sich, indem der Widerstand mit dem Strom multipliziert wird. Diese Gleichung ist aufgrund der einfachen Arithmetik sehr leicht umzustellen, und schon läßt sich mit einer Division des Stromwertes durch die Spannung der Widerstand bzw. durch Teilen der Spannung durch einen Widerstandswert die dazugehörige Stromstärke ermitteln.

Das Ohm'sche Gesetz

In der Elektrotechnik werden diese Größen oft symbolisch geschrieben. Dabei steht U für die Spannung, I für den Strom und der Buchstabe R für den Widerstand. Ohms Gleichungen sehen dann so aus:

$$U=R \times I$$
$$R=U/I$$
$$I=U/R$$

Zum Ausrechnen der Leistung P wird noch die Formel

$$P=U \times I$$

gebraucht. Sie gibt den Stromverbrauch einer Schaltung bzw. eines Gerätes in Watt an, der sich aus der anliegenden Spannung multipliziert mit dem fließenden Strom ermitteln läßt.

Die Gleichung für die Leistung können Sie auch im Haushalt einsetzen, wenn Sie zum Beispiel den Wert einer neuen Sicherung berechnen wollen: Bei einer Waschmaschine mit einem Verbrauch von 3 kW bei 220 Volt rechnen Sie so: 3000 (Leistung in Watt) geteilt durch 220 (Netzspannung in Volt) ergibt einen Wert von 13,64 Ampere für den Strom. Sie müssen also eine Sicherung mit 16 Ampere einsetzen – das ist der nächsthöhere erhältliche Wert. Der betreffende Stromkreis kann dann – außer mit der Waschmaschine – noch mit insgesamt 520 Watt belastet werden (16 A – 13,64 A = 2,36 A x 220 Volt). In der nächsten Folge zeigen wir, wie Sie die Ohm'schen Gesetze experimentell nachvollziehen können.

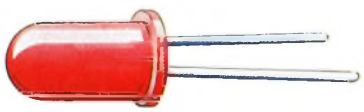
ACHTUNG: Der Hersteller kann die Garantieleistung nach dem unbefugten Öffnen des Gehäuses verweigern!

Bauteile

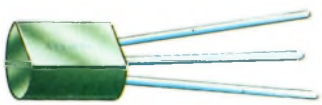
Für den Laien erscheint ein elektronischer Schaltplan recht kompliziert. Dabei kann die Darstellung eines Gerätes durch genormte Zeichen oft viel Arbeit sparen, weil sie Funktion und verwendete Bauteile übersichtlicher macht. Als Beispiel ist unten der Schaltplan einer Sprechanlage abgebildet. Daneben finden Sie die einzelnen Bauteile mit einer kurzen Beschreibung ihrer Funktionen. Auf Schaltplänen sind die Bauelemente meist mit einer codierten Bezeichnung – etwa „R1" oder „TR 2" – versehen. Die Verbindungslinien im Plan stellen die Leitungen dar – im Gerät sind es entweder Drähte oder Leiterbahnen auf einer Platine.

1 Schalter
Schalter gibt es in vielen verschiedenen Ausführungen, die zudem noch unterschiedliche Funktionen haben. Der Kontakt kann dauerhaft erhalten bleiben oder sich beim Loslassen wieder lösen.

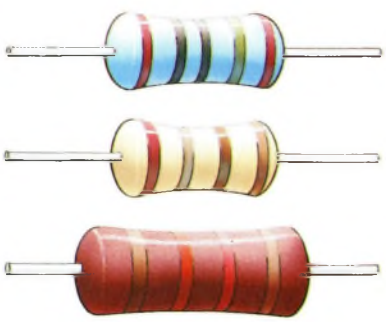
2 Leuchtdioden
Dioden sind die unkompliziertesten Halbleiter und stellen sozusagen ein „elektronisches Ventil" dar: Ein Strom kann sie nur in einer Richtung durchfließen. Manche Dioden haben kein Metall-, sondern ein durchsichtiges Kunstharz-Gehäuse. Bei Stromdurchgang geben diese „LEDs" Licht ab.

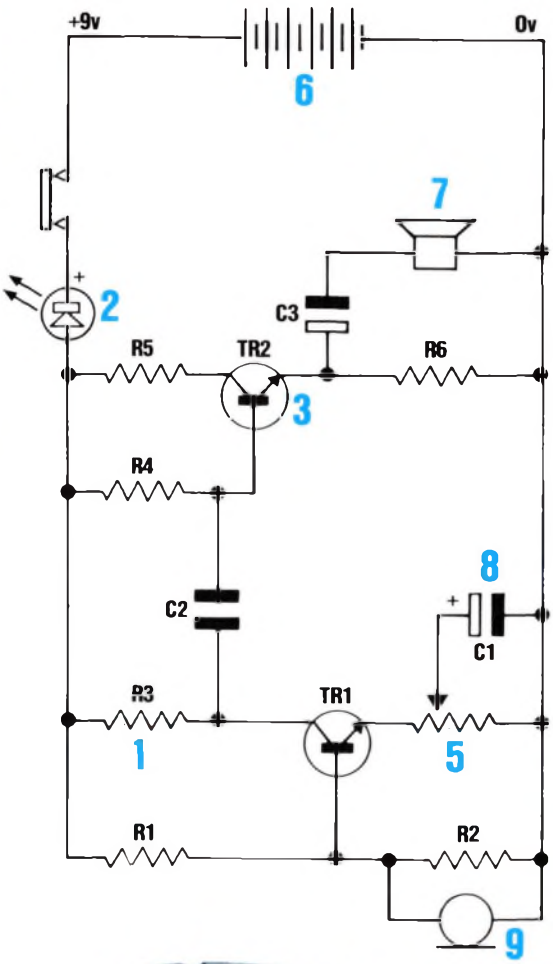


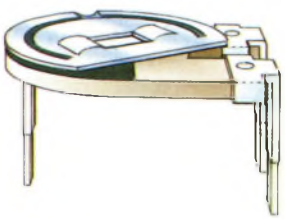
3 Transistoren
Je nach Konstruktion und Anschlußweise kann ein Transistor Schalter oder auch Verstärker sein. Die Erfindung des Transistors im Jahre 1947 leitete das Zeitalter der Microelektronik ein.



4 Widerstände
Durch Einbau von Material geringerer Leitfähigkeit wird der Stromfluß in einer Schaltung reguliert. Dazu dienen Widerstände in verschiedenen Größen, deren Widerstandswert sich an den farbigen Ringen ablesen läßt.



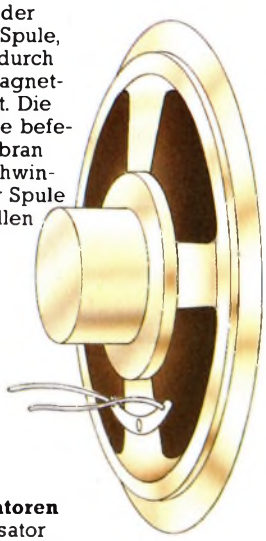




5 Einstellbare Widerstände
Widerstände, bei denen sich die Leitfähigkeit regulieren läßt, werden „Potentiometer" oder „Trimmer" genannt. Meist wird zum Einstellen ein Schleifer über eine Kohlebahn bewegt. Je nach Stellung des Schleifers wird der Widerstand größer oder kleiner.

6 Batterien
Kleinere Schaltungen lassen sich am günstigsten mit Batterien versorgen, weil sie eine sehr stabile Gleichspannung liefern.

7 Lautsprecher
Lautsprecher sind in ihrer Funktionsweise Mikrofonen sehr ähnlich. Im Lautsprecher durchfließt der Strom eine Spule, die sich dadurch in einem Magnetfeld bewegt. Die an der Spule befestigte Membran setzt die Schwingungen der Spule in Schallwellen um.



8 Kondensatoren
Ein Kondensator kann elektrische Ladungen speichern. Wenn die Anschlüsse mit einer Stromquelle verbunden werden, lädt sich der Kondensator auf. Er wird erst wieder entladen, wenn sich die Anschlüsse berühren oder durch einen externen Schalter miteinander verbunden werden.

9 Mikrofone
In einem Mikrofon werden die Schallwellen der Luft von einer feinen Membran aufgefangen. Eine damit verbundene Spule erzeugt bei ihren Bewegungen im Magnetfeld eine Spannung.

Bauteile

Jedes elektronische Gerät besteht aus einer Vielzahl von Bauteilen. Bei Microprozessoren oder Videochips zum Beispiel müssen zahlreiche Funktionen auf kleinstem Raum zusammengefaßt werden.

Auch das komplizierteste Bauelement besteht meist aus einer Vielzahl von einfachen Teilen. Diese Grundbausteine werden in der Elektrotechnik zwei verschiedenen Gruppen zugeordnet: Es gibt „aktive" und „passive" Bauteile.

Die passiven Bauelemente sind die einfacheren: Sie hemmen den Fluß elektrischer Signale, wobei unterschiedlichen Signalen ein jeweils anderer Widerstand entgegengesetzt wird. Ein Beispiel dafür sind Widerstände und Kondensatoren.

Aktive Komponenten können ein wenig mehr: Sie modifizieren das Ur-Signal. So zum Beispiel der Transistor – ihm wird ein schwaches Signal zugeführt, das er verstärkt wieder abgibt.

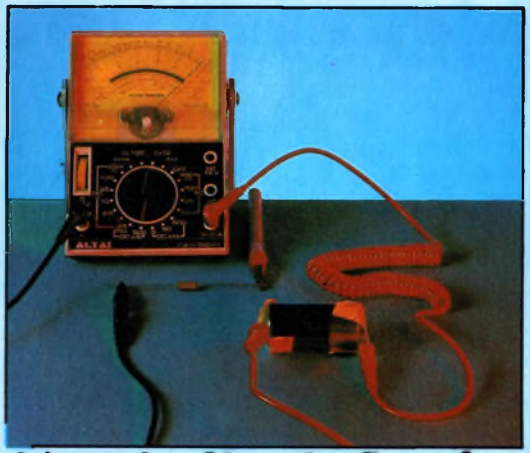
Die Halbleiter

Passive Bauteile bestehen aus einfachen Werkstoffen. Abgesehen vom Isoliermaterial ist ein Widerstand etwa aus Kupfer, Stahl und Kohle aufgebaut – alles leitfähige Elemente. Dagegen bestehen die aktiven Bauteile fast alle aus Germanium oder Silizium. Diese Elemente haben eine besondere Eigenschaft: Anders als etwa ein Metall leiten sie nur unter ganz bestimmten Bedingungen. Sie werden daher als „Halbleiter" bezeichnet.

Wegen ihres Aufbaus aus halbleitendem Material haben die aktiven Bauteile eine etwas ungewohnte Funktion: Wird zum Beispiel eine bestimmte Spannung über einem Halbleiter angelegt, kann man daraus noch nicht den fließenden Strom berechnen. Die passiven Bauteile machen es unserem Verständnis da etwas leichter, weil sich ihr Verhalten als leitfähiges Material aus den Ohmschen Gesetzen ermitteln läßt.

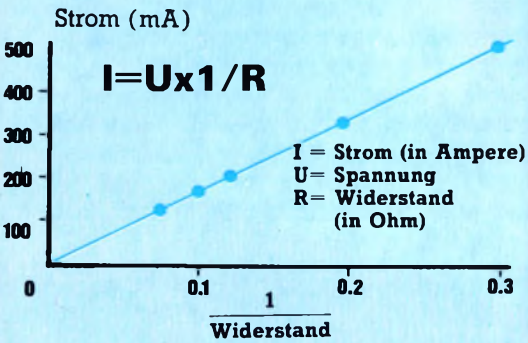
Für den Computer-Bereich ist der Transistor besonders wichtig, weil er sowohl für die Speicherung als auch für die Bearbeitung von Informationen gebraucht und deshalb entsprechend eingesetzt wird.

In der nächsten Folge wird gezeigt, wie man die Funktionen einfacher Logik-Gatter mit Hilfe von Transistoren und einer Steckleiste problemlos nachbauen kann.



Stimmt das Ohmsche Gesetz?

Mit etwas Draht, ein paar Widerständen, einer Batterie und einem Meßgerät läßt sich das ganz leicht feststellen: Die Widerstände sollten Werte zwischen 3,3 und 15 Ohm aufweisen und mit 1 Watt belastbar sein. Stellen Sie das Meßgerät auf Strommessung bis ca. 400 mA ein. Der Stromkreis wird nun wie oben im Bild geschlossen. Setzen Sie nacheinander die verschiedenen Widerstände ein und lesen jeweils das Meßgerät ab. Wenn Sie Strom und Kehrwert des Widerstandes in das Koordinatensystem eintragen, ergibt sich eine gerade Linie, in der bei konstanter Spannung I genau proportional 1/R ist.



Dioden

Dioden können Sie sich als „elektrisches Ventil" vorstellen. In der einen Durchgangsrichtung haben sie für den Strom nur einen ganz geringen Widerstand von wenigen Ohm. Mehrere Millionen Ohm setzt die Diode dem Stromfluß in der anderen Richtung entgegen. Dioden gehorchen dem Ohmschen Gesetz nicht – sie sind aus Halbleitermaterial aufgebaut. Der Ring auf dem Gehäuse gibt die Durchlaßrichtung an.

Spectrum

```
 20 DIM b$(3,7)
 25 DIM c$(4,7)
 30 INPUT "input colour of bands"
 40 INPUT "band number 1",b$(1)
 50 IF b$(1)="gold" OR b$(1)="silver" T
    HEN  GO TO 30
 60 INPUT "band number 2",b$(2)
 70 INPUT "band number 3",b$(3)
 80 PRINT
 90 FOR i=1 TO 3
100  RESTORE 180
110 FOR j=0 TO 9
120 READ c$(1),c$(2),c$(3),c$(4)
130 IF c$(i)=b$(i) THEN  GO SUB 1000
140 NEXT j
150 NEXT i
160 PRINT "  ohms"
170 STOP
180 DATA "black","0","0",".0","brown","
    1","1","0","red","2","2","00","
    orange","3","3",".000"
190 DATA "yellow","4","4",".0000","green
    ","5","5",".00000","blue","6","6","
    .000000"
200 DATA "violet","7","7",".0000000","gr
    ey","8","8",".0","white","9","9",".0"
1000 FOR k=1 TO 7
1020 IF c$(i+1),k)<>" " THEN  PRINT c$(
i+1),k);
1025 NEXT k
1030 RETURN
```

Acorn B

```
 10 REM resistor colour codes
 20 DIM B$(3),C$(3)
 30 PRINT "Input colour of bands:"
 40 INPUT "BAND NUMBER 1",B$(1)
 50 IF B$(1)="GOLD" OR B$(1)="
    SILVER" THEN GOTO 30
 60 INPUT "BAND NUMBER 2",B$(2)
 70 INPUT "BAND NUMBER 3",B$(3)
 80 PRINT
 90 FOR I=1 TO 3
100   RESTORE
110   FOR J=0 TO 9
120     READ C$(0),C$(1),C$(2),C$(3)
130     IF C$(0)=B$(I) THEN PRINT C$(I);
140   NEXT J
150   NEXT I
160 PRINT "  ohms"
170 END
180 DATA BLACK,0,0,, BROWN,1,1,0,
    RED,2,2,00, ORANGE,3,3,000
190 DATA YELLOW,4,4,0000, GREEN,5,5,
    00000, BLUE,6,6,000000
200 DATA VIOLET,7,7,0000000, GREY,8,8,,
    WHITE,9,9,,
```

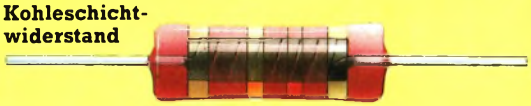
Beim Commodore 64 ersetzen Sie die Zeilen 40, 60 und 70 durch:

```
40 INPUT "BAND
   NUMMER 1 ";B$(1)
60 INPUT "BAND
   NUMMER 2 ";B$(2)
70 INPUT "BAND
   NUMMER 3 ";B$(3)
```

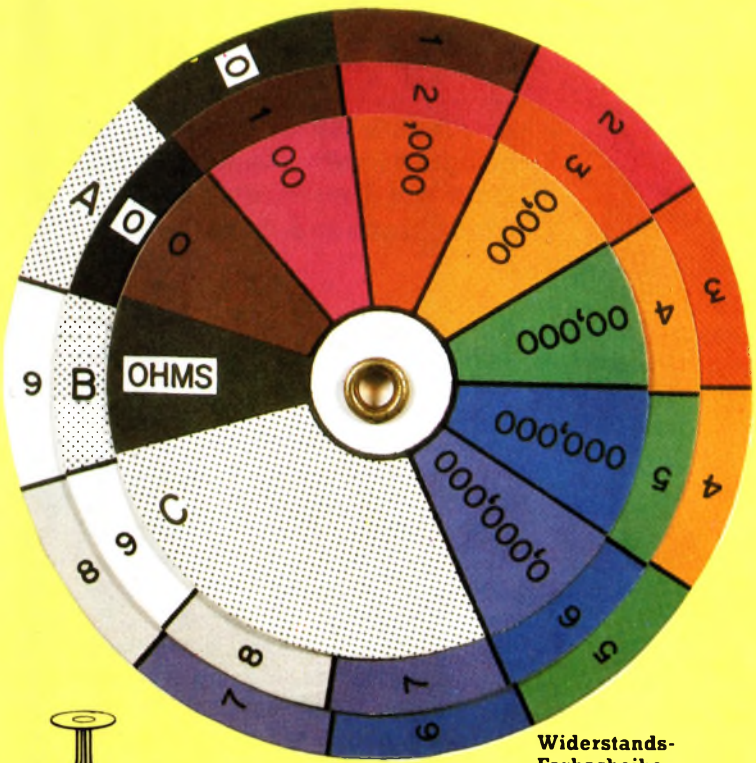
Widerstände

Gewickelter Widerstand

Kohleschicht-widerstand

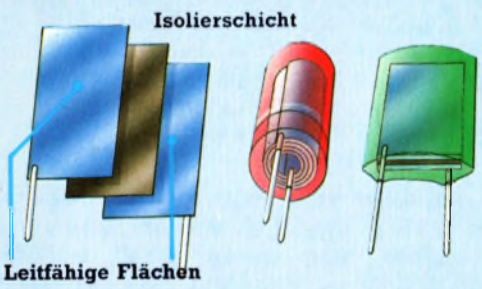


Widerstände sind einfachste elektronische Bauelemente. Bei gewickelten Widerständen ist ein Widerstandsdraht um einen Kern herum aufgerollt und mit einer Isolierschicht abgedeckt. Der Strom muß durch die gesamte Drahtlänge hindurchgehen und wird dabei „gebremst". Kohleschichtwiderstände arbeiten ähnlich.
Beide Widerstands-Typen werden durch farbige Ringe gekennzeichnet, von denen sich der jeweilige Wert des Bauteils ablesen läßt. Die zwei ersten Ringe bestimmen den Zahlenwert, der mit dem Wert des dritten Rings multipliziert den Widerstand angibt. Silberne und goldene Ringe definieren die Genauigkeit und zeigen, von welcher Seite her die Ringe gelesen werden – man liest immer auf die silbernen und goldenen Ringe zu. Damit Sie die Farben nicht auswendig lernen müssen, können Sie das nebenstehende Programm zur Bestimmung eines Widerstandes verwenden.

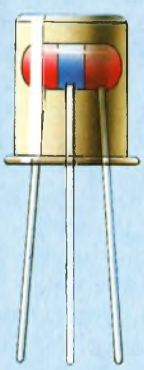


Widerstands-Farbscheibe
Zum schnellen Erkennen eines Widerstandes können Sie sich eine Farbscheibe herstellen. Stellen Sie einfach die Farben übereinander und lesen Sie den Wert ab.

Kondensatoren



Kondensatoren sind Widerstände für Wechselspannungen. Wechselspannungen hoher Frequenz gehen leichter durch sie hindurch als solche mit niedriger Frequenz. Man verwendet sie daher häufig zum Ausfiltern bestimmter Frequenzen. Kondensatoren bestehen aus zwei leitfähigen Flächen, die durch eine Isolierschicht getrennt sind. Je nach der erforderlichen Spannungsfestigkeit wird die Isolierschicht aus speziellen Salzen oder aus Keramik hergestellt. Um die Kondensatoren möglichst kompakt zu halten, werden die drei Schichten meist aufgewickelt und mit Isoliermasse vergossen.



Transistoren

Mit der Erfindung des Halbleiterelementes „Transistor" begann das Zeitalter der Elektronik – von allen vorgestellten Bauteilen ist der Transistor das komplizierteste. Transistoren können auf zweifache Weise eingesetzt werden: einmal als Verstärker, der einen geringen Eingangsstrom in einen hohen Ausgangsstrom umwandelt, zum anderen als Schalter, mit dem sich der Stromfluß elektronisch unterbrechen und fortsetzen läßt. Auf dieser Fähigkeit basiert die digitale Elektronik.
Transistoren bestehen wie Dioden aus halbleitendem Material. Sie haben jedoch insgesamt drei Anschlüsse, die als Basis, Kollektor und Emitter bezeichnet werden. Ob der Transistor als Schalter oder als Verstärker arbeitet, hängt von der Position seiner Anschlüsse in der Gesamtschaltung ab. Bei Transistoren können Sie nicht ohne weiteres erkennen, welcher Anschluß nun Basis, Emitter oder Kollektor ist – das variiert jeweils bei den unterschiedlichen Transistoren. Wer es genau wissen will, braucht eine Transistor-Vergleichstabelle.